

Politechnika Wrocławskiego
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka Techniczna (ITE)**
Specjalność: **Inżynieria Systemów Informatycznych (INS)**

**PRACA DYPLOMOWA
MAGISTERSKA**

Analiza porównawcza jakości klasyfikatorów danych tabelarycznych w systemach wykrywania intruzów w sieciach komputerowych

inż. Bartosz Błyszcz

Opiekun pracy
dr inż. Tomasz Babczyński

Słowa kluczowe: 3-6 słów

WROCŁAW, 2024

Streszczenie

Wykaz skrótów

Skrót	Nazwa angielska	Nazwa polska
GA	<i>Genetic Algorithm</i>	Algorytm Genetyczny
GP	<i>Genetic Programming</i>	Programowanie Genetyczne
IDS	<i>Intrusion Detection System</i>	System wykrywania intruzów
IPS	<i>Intrusion Prevent System</i>	System zapobiegania włamaniom
GNB	<i>Gaussian Naive Bayes</i>	Naiwny Klasyfikator Bayesa wykorzystujący rozkład Gaussa
ANN	<i>Artificial Neural Network</i>	Sztuczna sieć neuronowa
CNN	<i>Convolutional Neural Network</i>	Konwolucyjna sieć neuronowa
ML	<i>Machine Learning</i>	Uczenie maszynowe
AI	<i>Artificial Intelligence</i>	Sztuczna Inteligencja
IDS	<i>Intrusion Detection System</i>	System Wykrywania Intruzów
SVM	<i>Support Vector Machine</i>	Maszyna Wektorów Nośnych
AUC	<i>Area Under Roc Curve</i>	Przestrzeń pod krzywą ROC
LCDPs	<i>Low-code Development Platforms</i>	Platforma Low-code
BI	<i>Business Intelligence</i>	Narzędzia biznesowe do przekształcania danych
CDN	<i>Content Delivery Network</i>	Sieć dostarczania zawartości
MLP	<i>Multilayer Perceptron network</i>	Sieć wielowarstwowa perceptronowa
Azure ML	<i>Azure Machine Learning Studio</i>	
GAGNB	<i>Gaussian Naive Bayes - with GA</i>	
HTTPS	<i>Hyper Text Transfer Protocol Secure</i>	Protokół służący do komunikacji w sieci internetowej
REST	<i>Representative State Transfer</i>	Rozwiązanie architektoniczne służące do komunikacji w sieci internetowej

Spis treści

1. Wstęp.....	10
1.1. Wprowadzenie i uzasadnienie tematu pracy	10
1.2. Cel i zakres pracy.....	11
2. Podsumowanie pracy inżynierskiej	12
2.1. Algorytm genetyczny	12
2.2. Klasyfikator Naiwny Bayesa.....	13
2.3. Systemy wykrywania intruzów.....	13
2.4. Autorski algorytm opracowany w ramach pracy inżynierskiej.....	14
3. Sztuczna inteligencja	16
3.1. Uczenie maszynowe.....	16
3.1.1. Uczenie nienadzorowane	17
3.1.2. Uczenie nadzorowane	18
3.1.3. Uczenie częściowo nadzorowane	18
3.1.4. Uczenie przez wzmacnianie	18
3.2. Sztuczna sieć neuronowa	19
3.3. Głębokie uczenie	21
4. Klasyfikacja danych	23
4.1. Metryki klasyfikacji danych.....	23
4.1.1. Dokładność	24
4.1.2. Precyza.....	24
4.1.3. Czułość.....	24
4.1.4. Swoistość	25
4.1.5. F1	25
4.1.6. ROC - AUC	25
5. Platformy low-code/no-code	27
5.1. Microsoft PowerApps	28
5.2. Amazon QuickSight.....	29
5.3. Google AppSheet.....	30
6. Microsoft Azure	31
6.1. Infrastruktura	32
6.2. Machine Learning Studio	33
7. Doświadczenie	35
7.1. Metodologia badawcza	35
7.2. Założenie techniczne	36

7.3.	Dane	36
7.4.	Algorytmy wykorzystane w doświadczeniu	37
7.4.1.	Two-Class Support Vector Machine	37
7.4.2.	Two-Class Boosted Decision Tree	39
7.4.3.	Two-Class Decision Forest.....	41
7.4.4.	Two-class Neural Network	41
7.4.5.	Two-Class Average Perceptron.....	42
7.4.6.	Gausian Naive Bayes - with GA	43
7.4.7.	DANet	44
7.5.	Programistyczne środowisko badawcze	46
7.6.	Przebieg eksperymentu	48
7.6.1.	Przygotowanie danych.....	48
7.6.2.	Trenowanie oraz testowanie algorytmów.....	49
7.6.3.	Utworzenie tabeli porównawczej	50
7.6.4.	Wyniki danych treningowych.....	51
7.7.	Wyniki danych testowych	53
7.8.	Analiza uzyskanych wyników.....	57
7.8.1.	Wyniki dopasowania	57
7.8.2.	Wyniki precyzji.....	58
7.8.3.	Wyniki czułości.....	59
7.8.4.	Wyniki F1	60
7.8.5.	Wyniki AUC	61
7.9.	Analiza wyników	62
8.	Podsumowanie	66

Wykaz rysункów

2.1	Schemat algorytmu genetycznego	12
2.2	Klasyfikacja zbioru danych: Monday-WorkingHours	15
3.1	Graficzne przedstawienie podziałów sztucznej inteligencji	16
3.2	Podział uczenia maszynowego	17
3.3	Uczenie nienadzorowane	17
3.4	Uczenie nadzorowane	18
3.5	Uczenie przez wzmacnianie	19
3.6	Schemat sieci neuronowej	20
3.7	Schemat pojedynczego neuronu - perceptronu	21
3.8	Schemat prostej głębskiej sieci neuronowej	22
4.1	Przykład krzywej ROC	25
5.1	Wordpress.com	27
5.2	PowerApps od Microsoft	28
5.3	Amazon QuickSight	29
5.4	Google AppSheet	30
6.1	Schemat podziału usług MS Azure	32
6.2	Zdjęcia z mapy infrastruktury Microsoft Azure	33
7.1	Schemat SVM	38
7.2	Potok zadań dla modelu <i>Two-Class Support Vector Machine</i>	38
7.3	Sposób działania algorytmu	39
7.4	Potok zadań dla modelu <i>Two-Class Boosted Decision Tree</i>	40
7.5	Potok zadań dla modelu <i>Two-Class Decision Forest</i>	41
7.6	Potok zadań dla modelu <i>Two-Class Neural Network</i>	42
7.7	Potok zadań dla modelu <i>Two-Class Average Perceptron</i>	43
7.8	Potok zadań dla modelu	44
7.9	Sposób działania DANet	45
7.10	Potok zadań dla modelu <i>DANet</i>	45
7.11	Potok zadań	47
7.12	Potok normalizacji danych	48
7.13	Potok zadań dla algorytmów klasyfikacyjnych	49
7.14	Potok zadań dla algorytmów klasyfikacyjnych	50
7.15	Moduły odpowiedzialne za przetworzenie wyników	50

7.16 Wyniki testów algorytmów klasyfikacyjnych na danych treningowych	52
7.17 Wyniki testów algorytmów klasyfikacyjnych na danych testowych dla próby 1	54
7.18 Wyniki testów algorytmów klasyfikacyjnych na danych testowych dla próby 2	55
7.19 Wyniki testów algorytmów klasyfikacyjnych na danych testowych dla próby 3	56
7.20 Dokładność algorytmów	58
7.21 Precyzaja algorytmów	59
7.22 Czułość algorytmów	60
7.23 F1 algorytmów	61
7.24 AUC algorytmów	62

Wykaz tabel

2.1	Klasyfikacja zbioru danych: Monday-WorkingHours	14
4.1	Macierz pomyłek	23
7.1	Metodologia badawcza	36
7.2	Założenia techniczne pracy dyplomowej	36
7.3	Liczba elementów należących do danej klasy w zbiorze treningowym	51
7.4	Liczba elementów należących do danej klasy w zbiorze testowym	53
7.5	Wynik dopasowania algorytmów.	57
7.6	Wynik precyzji algorytmów.	58
7.7	Wynik czułości algorytmów.	59
7.8	Wynik F1 algorytmów.	60
7.9	Wynik AUC algorytmów.	61
7.10	Założenia wykorzystywane do analizy statystycznej danych	62
7.11	Tabela rozkładu wartości dla t-Studenta	63
7.12	Wyniki testu t Studenta dla poszczególnych prób testowych	64

1. Wstęp

1.1. Wprowadzenie i uzasadnienie tematu pracy

Klasyfikacja danych tabelarycznych jest trudnym zagadnieniem do analizy, z powodu powszechnego występowania bardzo dużej ilości nieuporządkowanych danych, które zawierają wiele cech. To proces organizowania danych w tabeli, w celu ich łatwiejszej analizy, interpretacji czy dalszego przetwarzania. Podczas takiej kategoryzacji danych ważne jest właściwe dobranie algorytmu, ze względu na typ danych. Przykładowo zbiór danych tekstowych można klasyfikować za pomocą jednokierunkowej sieci neuronowej, a obrazy za pomocą sieci konwolucyjnej.

Obecnie istnieje wiele różnych algorytmów do klasyfikacji danych tabelarycznych. Jednymi z popularniejszych są: regresja logistyczna (*ang. logistic regression*), drzewo decyzyjne (*ang. decision tree*), las losowy (*ang. random forest*), maszyna wektorów nośnych (*ang. support vector machine*), naiwny klasyfikator Bayesowski (*ang. Naive Bayes classifier*). Przy wykorzystaniu tych algorytmów do kategoryzacji, ważne jest właściwe wybranie algorytmu, czyli rozpoznanie z jakimi danymi mamy do czynienia oraz porównanie wyników klasyfikacji, w celu wybrania najlepszego dopasowania.

Dane tabelaryczne występują w każdej dziedzinie, duże zestawy danych można spotkać w medycynie, nauce czy w finansach. Rosnąca liczba danych oraz ich zmienna struktura wymaga opracowywania coraz lepszych algorytmów klasyfikacji. Jednakże wyjątkowość danych sprawia, że trudno opracować uniwersalny algorytm klasyfikacji. Wiele rozwiązań jest tworzonych dla konkretnej struktury danych, co powoduje niemożność ich wykorzystania dla innych danych.

Przy rosnącej liczbie danych do analizy, rozwijają się metody ułatwiające ich klasyfikację. Coraz częściej wykorzystuje się rozwiązania z zakresu sztucznej inteligencji czy obliczeń chmurowych. Jednym z przykładów jest aplikacja *Machine Learning Studio*, dostępne w *Microsoft Azure*. Zawiera ona zestaw narzędzi, umożliwiających łatwiejsze kategoryzowanie danych czy tworzenie algorytmów klasyfikacji i ich porównywanie. Użycie chmury pozwala na wykorzystanie mocy obliczeniowej sklasteryzowanych jednostek wirtualnych do wykonywania obliczeń na odpowiednich maszynach wirtualnych czy do budowania skomplikowanych zautomatyzowanych procesów złożonych z wielu zadań (*ang. pipeline*). Natomiast wykorzystanie sztucznej inteligencji pozwala na wprowadzenie elementu uczenia się w celu lepszego rozpoznawania danych.

Zastosowanie tych narzędzi umożliwia automatyzację procesu badawczego, porównanie wyników działania różnych algorytmów oraz znaczne przyspieszenie badań. Ma to znaczenie przy rosnącym zapotrzebowaniu na analizę dużych zestawów danych.

W dobie rozwijających się hurtowni danych oraz sztucznej inteligencji coraz więcej firm przetrzymuje ogromne zbiory danych w sieci komputerowej. Dane takie mogą być poufne

bądź o znaczeniu strategicznym. Wyciek takich danych może mieć negatywne konsekwencje dla właścicieli danych bądź osób, których dane są przechowywane. Przykładem może być dostęp do prywatnych kont bankowych albo danych medycznych. Dlatego też dane te są zabezpieczane m.in. przy użyciu kryptografii. Pozwala to na zaszyfrowanie przesyłu danych poufnych jak i samych przesyłanych danych. Jednakże osobom planującym kradzież konkretnych danych dużo bardziej zależy na uzyskanie nieautoryzowanego dostępu do komputerów mogących posiadać dostęp do danych wrażliwych. Dzieje się tak, ponieważ dostęp do urządzeń, pozwala nagląd nie tylko do konkretnych danych ale i do całej sieci intranetowej np. banku. Dostęp taki może zostać również wykorzystany do np. wgrania szkodliwego oprogramowania umożliwiającej założenie „tylnej furtki” (ang. *backdoor*). W celu ograniczenia możliwych nieautoryzowanych dostępów z zewnątrz powstało oprogramowanie *IDS, IPS* (ang. *Intrusion Detection system, Intrusion Prevent System*). Są to systemy do ostrzegania i przeciwdziałania atakom na sieć komputerową [1].

Dane sieciowe są bardzo złożone oraz posiadają dużo cech, dlatego do ich analizy można wykorzystać algorytmy uczenia maszynowego pozwalające na wykrycie nietypowego ruchu sieciowego. Klasyczne systemy IDS bazują głównie na regułach wykluczających konkretny ruch sieciowy. Przewagą uczenia maszynowego jest to, że może wykrywać anomalie niewchodzące w skład reguł bezpieczeństwa w sieci. Może to umożliwić przy niewielkim koszcie wytrenowania sieci wczesne i szybsze wykrywanie potencjalnych ataków niż w przypadku klasycznych systemów IDS opartych o reguły.

1.2. Cel i zakres pracy

Celem niniejszej pracy dyplomowej jest ocena jakości opracowanego w pracy inżynierskiej autorskiego sposobu klasyfikacji danych tabelarycznych, wykorzystującego algorytm genetyczny oraz klasyfikator Naiwny Bayesa. W tym celu dokonano analizy porównawczej rozwiązania wraz z algorytmami dostępnymi w aplikacji *Machine Learning Studio*. Algorytmy opisano w Podrozdziale 7.4.

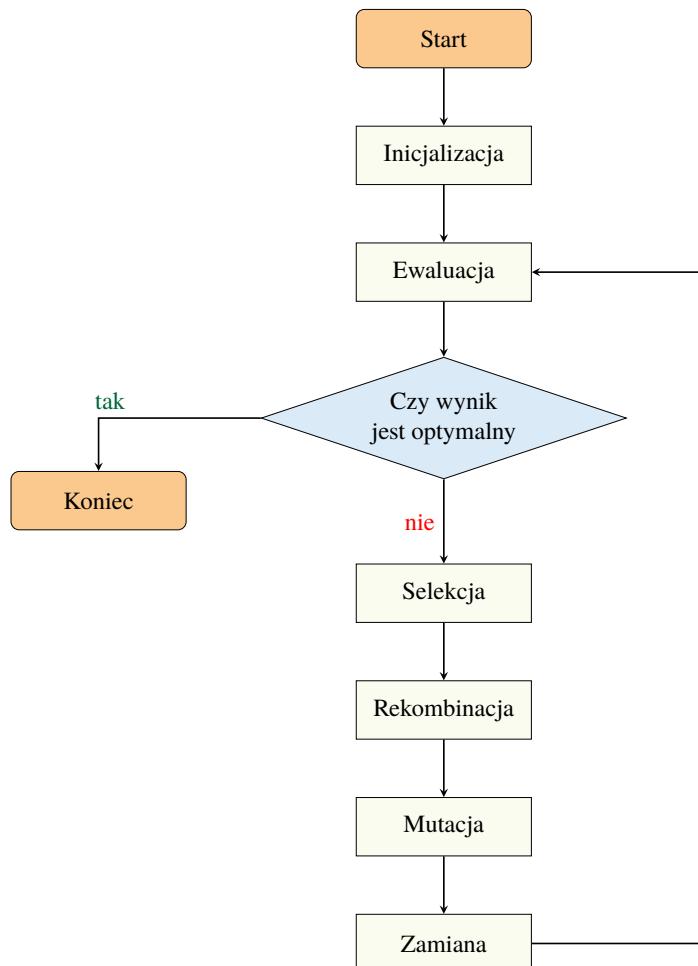
Praca składa się z 2 części. W części teoretycznej (Rozdziały 2 - 5) dokonano przeglądu dostępnych rozwiązań chmurowych (podejścia low-code/no-code). Oprócz tego opisano zagadnienia związane ze sztuczną inteligencją. W części badawczej (Rozdziały 6 - 8) przygotowano programistyczne stanowisko badawcze. W tym celu scharakteryzowano metryki jakościowe i opracowano eksperyment. Wykonano analizę porównawczą i statystyczną otrzymanych wyników. Dane wykorzystane do badań pochodziły z Instytutu Cyberbezpieczeństwa, działającego przy Uniwersytecie Nowy Brunszwik.

2. Podsumowanie pracy inżynierskiej

W ramach pracy inżynierskiej opracowano autorski algorytm *Gaussian Naive Bayes - with GA (GAGNB)*, wykorzystujący algorytm genetyczny oraz klasyfikator Naiwny Bayesa. Do badań wykorzystano dane uzyskane z Instytutu Cyberbezpieczeństwa, działającego przy Uniwersytecie Nowy Brunszwik. Dane użyto też w niniejszej pracy dyplomowej. Zbiór danych został opisany w **sekcji 7.3**.

2.1. Algorytm genetyczny

Algorytm genetyczny, który został zastosowany w GAGNB, to algorytm, który przeszukuje przestrzeń alternatywnych rozwiązań dla danego problemu, by znaleźć najlepszy wynik [2]. Działanie algorytmu genetycznego jest inspirowane m.in. ewolucją biologiczną, stąd też pochodzą nazwy kolejnych kroków, które zostały przedstawione na **rysunku 2.1**.



Rys. 2.1. Schemat algorytmu genetycznego
Źródło: Opracowanie własne na podstawie: [1–3]

Poniżej opisano przebieg algorytmu genetycznego. Proces składa się z 6 etapów:

1. **Inicjalizacja** - metoda inicjująca populację losowych osobników, wykorzystywanych podczas obliczeń;
2. **Ewaluacja** - etap sprawdzenia jakości wygenerowanych danych;
3. **Selekcja** - część algorytmu skupiająca się na wybraniu n najlepszych rozwiązań, które przejdą do następnej populacji;
4. **Rekombinacja** - element tworzący nową populację na bazie poprzedniej populacji. Wykorzystuje do tego mechanizm krzyżowania rodziców;
5. **Mutacja** - metoda wprowadzająca zmianę w n losowych miejsc w genomie;
6. **Zamiana** - miejsce wymiany starej populacji na nową [1–3].

2.2. Klasyfikator Naiwny Bayesa

Autorski algorytm GAGNB w etapie ewaluacji wykorzystuje naiwny klasyfikator Bayesa oparty o rozkład normalny Gaussa (*ang. Gaussian Naive Bayes, GNB*). Równanie klasyfikatora oparte jest o twierdzenie Bayesa. Opisuje ono prawdopodobieństwo wystąpienia zdarzenia na podstawie znajomości warunków zdarzenia [4]. Twierdzenie zostało przedstawione na **równaniu 2.1**.

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \quad (2.1)$$

gdzie:

- H oraz X są różnymi zdarzeniami;
- $P(X) \neq 0$;
- $P(H|X)$ - prawdopodobieństwo wystąpienia zdarzenia H jeśli zdarzenie X jest prawdziwe;
- $P(X|H)$ - prawdopodobieństwo wystąpienia zdarzenia X jeśli zdarzenie H jest prawdziwe;
- $P(H)$ oraz $P(X)$ to prawdopodobieństwa zaobserwowane, bez żadnych warunków [5].

Równanie wykorzystujące rozkład normalny Gaussa zostało przedstawione na **równaniu 2.2**. W kodzie wykorzystano bibliotekę scikit-learn [6]. Posiada ona implementację GNB.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{(x_i-\mu_k)^2}{2\sigma_k^2}} \quad (2.2)$$

gdzie:

- σ_k^2 - wariancja danych w kolumnie,
- σ_k - odchylenie standardowe danych w kolumnie,
- μ - średnia wartość w kolumnie [5].

2.3. Systemy wykrywania intruzów

Systemy wykrywania intruzów (*ang. Intrusion Detection System, IDS*) powstały w celu monitorowania sieci komputerowych. Ich zadaniem jest, rozpoznawanie szkodliwego ruchu

sieciowego, detekcja zagrożeń oraz reagowanie na występujące anomalie. Klasyczny model wykrywania intruzów zawiera:

- **źródła informacji** służące do określenia czy doszło do ataku,
- **moduł analizujący** przetwarzający zebrane dane,
- **moduł odpowiadający** reagujący w określony sposób na wykryte anomalie [1, 7, 8].

IDS dzielimy na dwie grupy rozróżnione na podstawie sposobu działania. Są to:

- **systemy oparte na hoście** (*ang. Host Base Intrusion Detection System, HIDS*) - służą do zbierania informacji o zdarzeniach systemowych i wykrywania luk w systemach komputerowych,
- **systemy oparte na sieci komputerowej** (*ang. Network Base Intrusion Detection System, NIST*) - służą do wykrywania szkodliwej aktywności w sieci [1, 7, 9].

2.4. Autorski algorytm opracowany w ramach pracy inżynierskiej

Celem algorytmu była redukcja wymiarowości danych tabelarycznych, by zwiększyć jakość klasyfikacji danych. Uzyskano to poprzez wyznaczenie najistotniejszych cech zbioru. Algorytm genetyczny zastosowano w celu przygotowania zbioru prawdopodobnie istotnych kolumn poprzez oznaczenie ich pozycji za pomocą ciągu cyfr 0 i 1. Kolumny oznaczone cyfrą 1 były brane pod uwagę w procesie ewaluacji za pomocą klasyfikatora Naiwnego Bayesa. Cały proces trwał maksymalnie 1000 iteracji lub do momentu uzyskania minimum 90% dopasowania. Wynikiem działania autorskiego algorytmu jest zbiór kolumn istotnych w procesie klasyfikowania. W celu sprawdzenia jakości własnego rozwiązania wykorzystano następujące metody statystyczne:

- **metoda ANOVA** - analiza wariancji,
- **współczynnik korelacji Pearsona** - współczynnik określający zależność liniową pomiędzy zmiennymi losowymi,
- **współczynnik korelacji rang Spearmana** - współczynnik korelacji rang Pearsona, dla danych po ustaleniu rang.

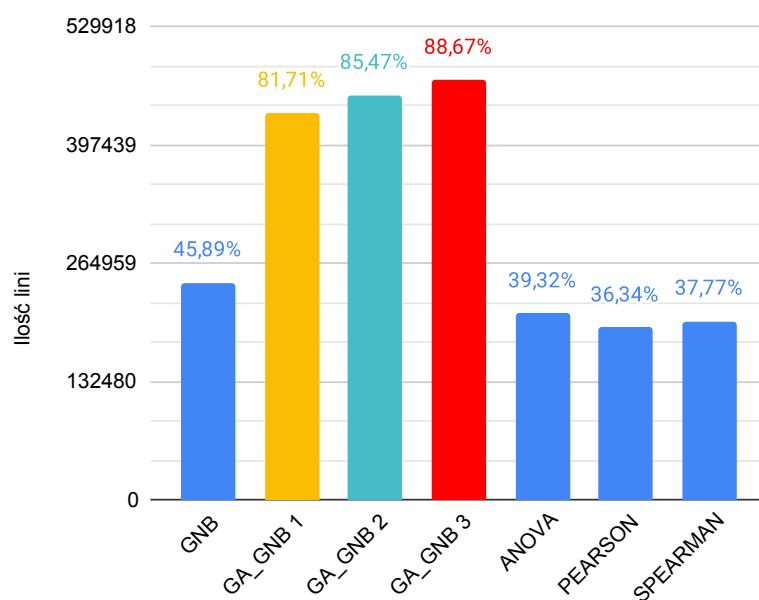
Analiza wykazała, że wyniki uzyskane za pomocą metod statystycznych były gorsze niż autorskiego algorytmu. Rezultaty badań przedstawia **tabela 2.1** oraz **wykres 2.2**.

Tabela 2.1. Klasyfikacja zbioru danych: Monday-WorkingHours
 Źródło: [1]

	Podstawowa	Klasyfikacja: Monday-WorkingHours				Zoptymalizowana Statystycznie		
		Zoptymalizowana GA			ANOVA	PEARSON	SPEARMAN	
		1	2	3				
Rozmiar danych [MB]	347,19	197,60	181,43	189,51			197,60	
Ilość linii [-]			529 918					
Czas operacji [s]	2,68	1,84	1,77	2,23	1,89	1,83	1,91	
Dokładność [%]	45,89	81,71	85,47	88,67	39,32	36,34	37,77	
Precyzja [%]			100					
Czułość [%]	45,89	81,71	85,47	88,67	39,32	36,34	37,77	
F1 [%]	62,91	89,94	92,16	93,99	56,45	53,31	54,83	
Zużycie pamięci [MB]	1853,44	980,06	885,42	932,74	980,06			

Wykres 2.2. przedstawia wyniki dokładności. Zastosowano następujące opisy:

- **GNB** - dopasowanie danych w podstawowym zbiorze;
- **GA_GNB 1** - dopasowanie danych w pierwszej próbie wykorzystania algorytmu autora;
- **GA_GNB 2** - dopasowanie danych w drugiej próbie wykorzystania algorytmu autora;
- **GA_GNB 3** - dopasowanie danych w trzeciej próbie wykorzystania algorytmu autora;
- **ANOVA** - dopasowanie danych z wykorzystaniem cech uzyskanych metodą ANOVA;
- **PEARSON** - dopasowanie danych z wykorzystaniem cech uzyskanych metodą współczynników korelacji Pearsona;
- **SPEARMAN** - dopasowanie danych w zbiorze zmodyfikowanym o rozwiązańie uzyskane metodą współczynników korelacji rang Spearmana.

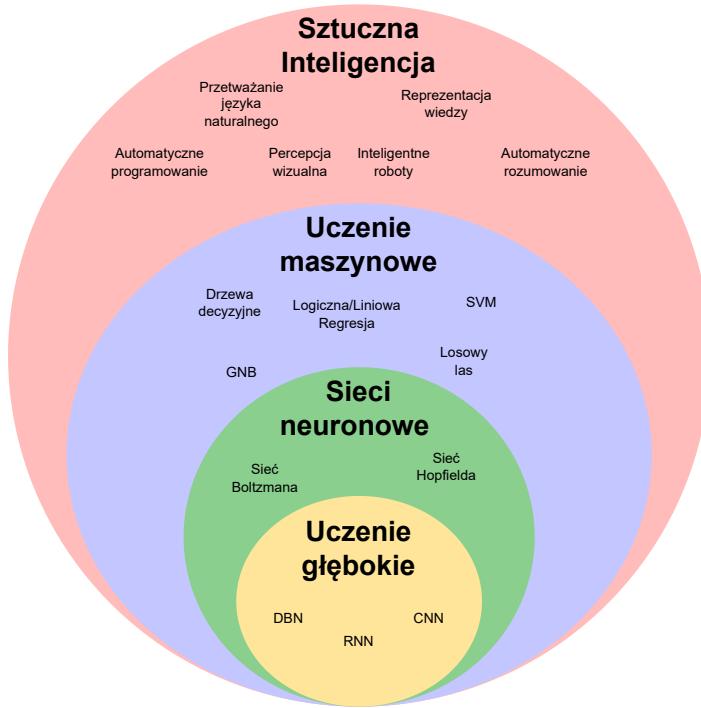


Rys. 2.2. Klasyfikacja zbioru danych: Monday-WorkingHours
 Źródło: [1]

Analizując powyższe wyniki można zauważyc, że najwyższe dopasowanie uzyskał autorski algorytm a najniższe - zbiór kolumn wyznaczonych metodą współczynników korelacji Pearsona. Świadczy to o wysokiej jakości, opracowanego w pracy inżynierskiej, algorytmu.

3. Sztuczna inteligencja

Słownik *Oxford English Dictionary* słowo „inteligencja” definiuje jako zdolność do rozumienia, analizy i dostosowania się do zmian [10]. Rozwój nauki i komputerów umożliwił badania nad tworem zbliżonym do możliwości ludzkiej inteligencji, czyli nad sztuczną inteligencją (ang. *Artificial Intelligence, AI*). W ostatnich latach SI dynamicznie się rozwija, znalazła zastosowanie w medycynie, finansach czy w badaniach naukowych. Wykorzystywana jest niemal na każdym etapie procesu badawczego: stawiania hipotez, budowania twierdzeń matematycznych, tworzenia i monitorowania badań, zbierania danych czy w wielu innych czynnościach [11, 12]. Wśród obszarów sztucznej inteligencji możemy wydzielić m.in. uczenie maszynowe, sieci neuronowe czy uczenie głębokie. **Rysunek 3.1** pokazuje rosnące zawężenie zagadnienia związanego z SI, gdzie uczenie głębokie jest najścislejszym a sztuczna inteligencja najszerzym.

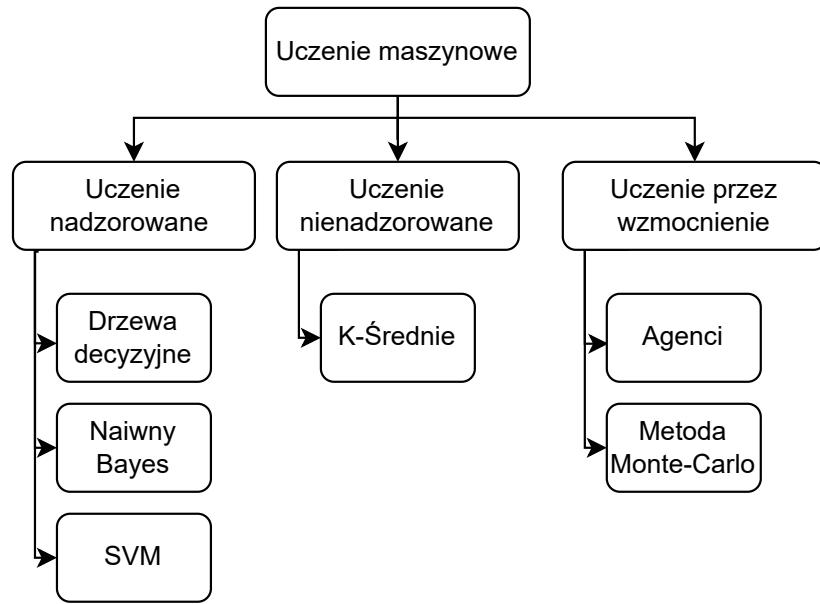


Rys. 3.1. Graficzne przedstawienie podziałów sztucznej inteligencji
Źródło: [13]

3.1. Uczenie maszynowe

Uczenie maszynowe (ang. *Machine Learning, ML*) to dziedzina nauki nad algorytmami oraz modelami statystycznymi, które pracują na dużych zbiorach danych, ucząc się je klasyfikować. Algorytmy ML wykorzystuje się m.in. w analizie danych, tworzeniu modeli predykcyjnych, rozpoznawaniu zdjęć czy mowy. Dodatkowo nie są zaprogramowane specyficznie pod konkretne zadanie, a jedynie pod grupę zadań. Uczenie maszynowe można podzielić na 3 typy, do których należą uczenie: nadzorowane, nienadzorowane oraz ze wzmacnieniem.

Podział pokazano na **rysunku 3.2**. Wykorzystanie konkretnego typu uczenia determinuje typ zadania, jaki ma być rozwiązyany [12, 13].

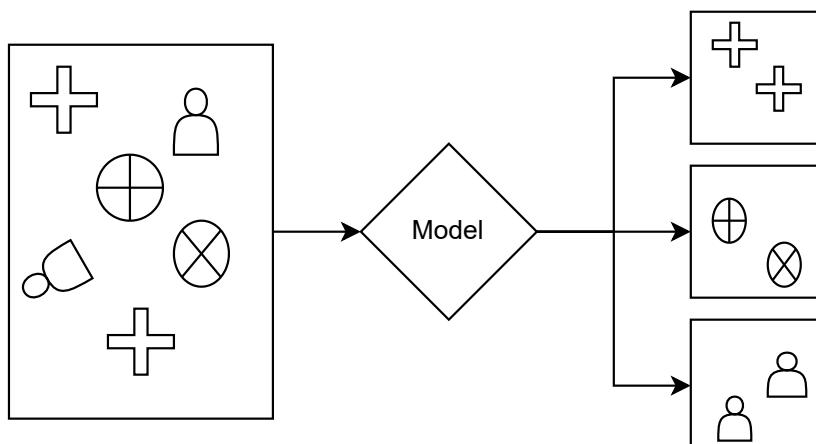


Rys. 3.2. Podział uczenia maszynowego

Źródło: [12]

3.1.1. Uczenie nienadzorowane

W przypadku uczenia nienadzorowanego algorytm ma sam znaleźć czy występują jakieś wzorce. W porównaniu do uczenia nadzorowanego, tu nie stosuje się zbioru oznaczonego. Metodę wykorzystuje się w pracy z danymi, których nie da się zdefiniować np. przy detekcji anomalii, szukaniu wzorców. Pozwala to m.in. na segmentację grup czy wykrywanie anormalnych zachowań np zwiększonego zużycia prądu spowodowanego uszkodzeniem urządzenia elektrycznego. Wśród algorytmów uczenia nienadzorowanego wyróżniamy dwie główne metody: K-średnie, klasteryzacja. Schemat uczenia nienadzorowanego poprzez klasteryzację jest pokazany na **rysunku 3.3** [11, 12].

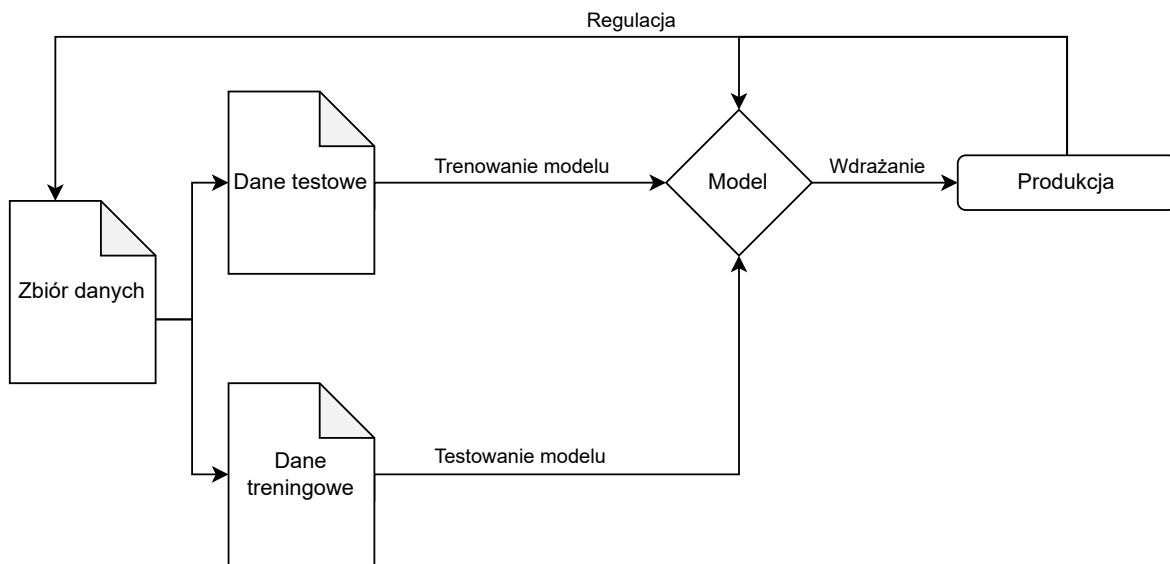


Rys. 3.3. Uczenie nienadzorowane

Źródło: Opracowanie własne

3.1.2. Uczenie nadzorowane

W procesie uczenia nadzorowanego stosuje się zbiór posiadający etykiety. Model uczy się przyporządkowywać określone cechy do konkretnych kategorii. Dane wejściowe dzielone są na dane treningowe i dane testowe. Zbiór treningowy jest wykorzystywany do trenowania modelu, a zbiór testowy do sprawdzenia rezultatu. W trakcie trenowania występuje regulacja modelu. Model uczenia został przedstawiony na **rysunku 3.4** [11, 12].



Rys. 3.4. Uczenie nadzorowane

Źródło: [12]

Algorytmy uczenia nadzorowanego można zastosować między innymi do weryfikacji ruchu sieciowego w celu określenia czy ruch bezpieczny, przez co można to zastosować w systemach wykrywania intruzów (ang. *Intrusion Detection System, IDS*). Algorytmy wchodzące w skład uczenia nadzorowanego to m.in. klasyfikator Naiwny Bayesa, drzewo decyzyjne, maszyna wektorów nośnych [11, 12].

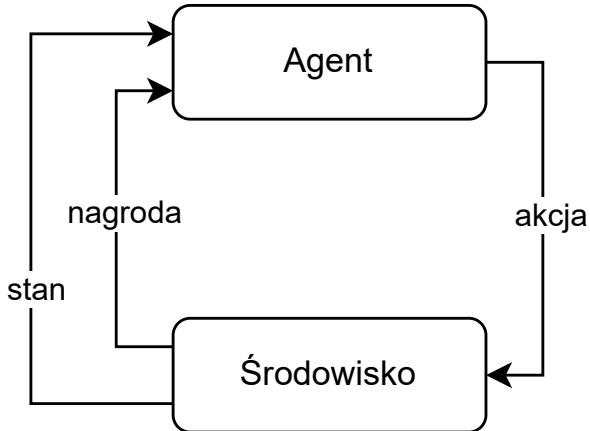
3.1.3. Uczenie częściowo nadzorowane

Jednym ze szczególnych przypadków połączenia uczenia nadzorowanego i nienadzorowanego jest uczenie częściowo nadzorowane. Wykorzystywane jest w sytuacjach, kiedy istnieje zbyt duży zbiór danych, którego oznaczenie wiążałoby się z zbyt dużymi kosztami czasowymi i finansowymi. Wtedy etykietyzowana jest mała próbka danych, która służy do utworzenia modelu. Na podstawie takiego modelu oznacza się pozostały zbiór danych. Takie rozwiązanie stosuje się m.in. w bankowości, klasyfikowaniu stron internetowych poprzez wyszukiwanie treści na stronie i kategoryzowaniu ich oraz wyznaczaniu trasy GPS [12, 14].

3.1.4. Uczenie przez wzmacnianie

Uczenie przez wzmacnianie jest najbliższym ludzkiemu uczeniu się. Zbliżona jest do metody „*kija i marchewki*”. To jest uczenie poprzez nagradzanie dobrych rozwiązań, a karanie złych. W odróżnieniu od pozostałych metod uczenia maszynowego, w uczeniu przez wzmacnianie nacisk kładziony jest na przygotowanie odpowiedniego środowiska. Jako środowisko określa

się zadania lub symulacje, z którymi agent wchodzi w interakcję. Schemat działania przedstawiono na **rysunku 3.5**. W tym typie uczenia nie są ważne dane, a rezultat.



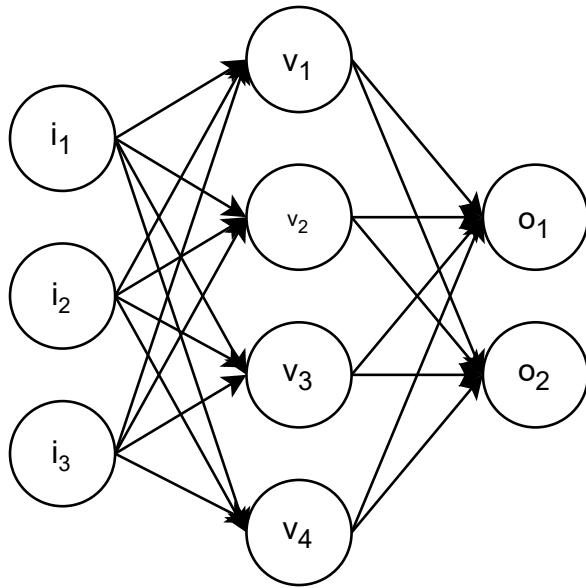
Rys. 3.5. Uczenie przez wzmocnienie
Źródło: [12]

Uczenie przez wzmocnienie jest wykorzystywane m.in. w trenowaniu pojazdów autonomicznych. Sztuczna inteligencja znajduje się np. w środowisku miejskim i zbiera dane z otoczenia. SI jest nagradzane za wybór lepszych tras do jazdy, czyli dróg asfaltowych zamiast polnych [11, 12].

3.2. Sztuczna sieć neuronowa

Ludzki mózg jest najbardziej złożonym organem znany ludziom. Składa się z wielu połączonych ze sobą komórek neuronowych, które przetwarzają równolegle wiele informacji. Ta struktura zainspirowała naukowców do zaimplementowania podobnych rozwiązań w komputerach. Przykładem tego są algorytmów, wchodzące w skład sztucznych sieci neuronowych (*ang. artificial neural network, ANN*), m.in. sieci Kohonena, sieci Hopfielda, sieci konwolucyjne. Sieci te podczas wykonywania np. klasyfikacji danych, próbują odwzorować działanie ludzkiego mózgu. Mimo tych osiągnięć symulacja ludzkiej świadomości oraz emocji wciąż jest jedynie w sferach fantazji [15].

Sztuczna sieć neuronowa to model uczenia maszynowego, który podejmuje decyzje w sposób zbliżony do ludzkiego mózgu. Budowa i działanie sieci jest opartne na działaniu ludzkich neuronów. ANN jest zbudowana z połączonych ze sobą warstw sztucznych neuronów. Sieć neuronowa posiada warstwę wejściową, warstwy ukryte i warstwę wyjściową. Każda warstwa składa się ze zbioru sztucznych neuronów. Przykładowa sieć została przedstawiona na **rysunku 3.6**.



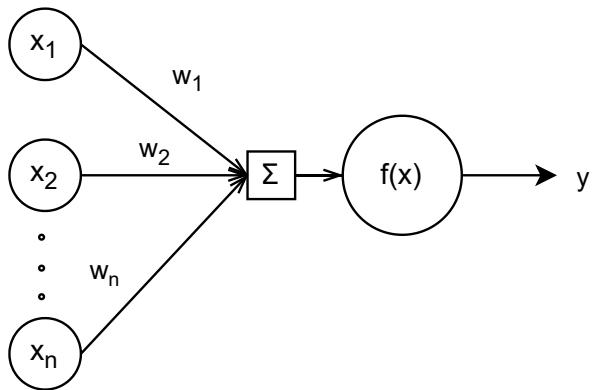
$i_x \forall x \in [1, 2, 3]$: dane wejściowe
 $v_x \forall x \in [1, 2, 3, 4]$: neurony w warstwie ukrytej
 $o_x \forall x \in [1, 2]$: dane wyjściowe

Rys. 3.6. Schemat sieci neuronowej
 Źródło: Opracowanie własne

Warstwa wejściowa za pomocą węzłów wejściowych przyjmuje i przetwarza dane wejściowe. Dane są analizowane i przeliczane, a następnie przekazywane do warstwy ukrytej. Sieci mogą posiadać dużą liczbę warstw ukrytych. Jeśli posiadają tych warstw znaczną liczbę, zaczyna się je określać jako głębokie sieci neuronowe. Głębokie sieci neuronowe zostały opisane w **sekcji 3.3**. Po przejściu danych przez warstwy ukryte trafiają do warstwy końcowej, a następnie zwracany jest wynik obliczeń. W warstwie wyjściowej może być n węzłów, gdzie $n \geq 1$. W przypadku klasyfikacji binarnej (0/1) otrzymamy jeden wynik, a w przypadku klasyfikacji wieloklasowej wyników będzie więcej.

Sieć ta potrafi się dostosowywać do danych wejściowych tak, aby uzyskać odpowiedni wynik. W tym celu wykonuje wtedy proces uczenia, stosując do tego na przykład algorytm wstecznej propagacji wag. Algorytm ten dostosowuje wagi, którymi opisano poszczególne połączenia do neuronu. Wagi nadają rangę wejścia, co oznacza, że im wyższa waga tym ważniejsze połączenie. W zależności od problemu istnieje wiele różnych sieci, które można zastosować. Jednym z bardziej skomplikowanych rzeczy w doborze sieci jest dobór warstw ukrytych oraz ilości neuronów, ponieważ dobór jest oparty o własną wiedzę i doświadczenie twórcy. Sieci neuronowe możemy podzielić na wiele rodzajów. Wśród nich możemy wyróżnić między innymi:

- **perceptron** - jest to najstarszy i najprostszy przykład sieci neuronowej złożonej z jednego perceptronu (neuronu). Został przedstawiony na **rysunku 3.7**. Stosuje się go jako punkt wyjściowy do tworzenia bardziej złożonych modeli ANN opartych o warstwy złożone z perceptronów. Może on też posłużyć do klasyfikacji binarnej, w której oczekujemy jako wyniku 1 albo 0.



Σ : sumator

$f(x)$: funkcja aktywacyjna

$w_x \forall x \in [1, 2, \dots, n]$: wagi

$x_x \forall x \in [1, 2, \dots, n]$: wejścia

Rys. 3.7. Schemat pojedynczego neuronu - perceptronu

Źródło: Opracowanie własne

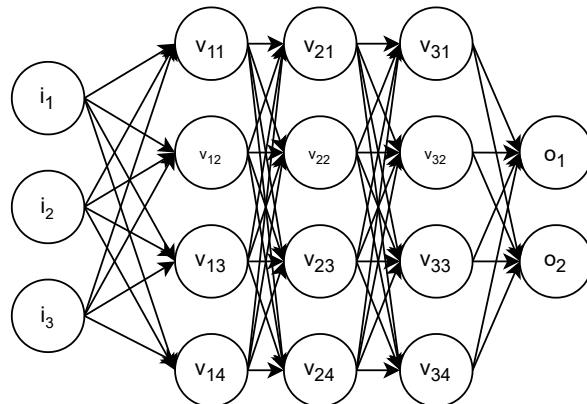
- **sieci wielowarstwowe perceptronowe** - jest to sieć złożona z wielu warstw połączonych ze sobą neuronów, najprostszy model sieci zaprezentowany na **rysunku 3.6**. Składa się z warstwy wejściowej, warstw (jednej bądź wielu) ukrytych oraz warstwy wyjściowej. W neurony w tej sieci w porównaniu do perceptronów, mają funkcję aktywacyjną sigmoidalną, ze względu na rozwiązywanie problemów nieliniowych (posiadających więcej rozwiązań niż dwa 0/1). Można je zastosować na przykład do klasyfikacji danych;
- **sieci konwolucyjne (CNN)** - są to sieci służące do rozpoznawania obrazów, nazwa wzięła się od wykonywanej na obrazie operacji konwolucji (splotu). Sieci te posiadają dodatkowe warstwy konwolucyjne oraz spłaszczenia, które pozwalają zamienić reprezentację obrazu w pojedyncze wartości;
- **sieci rekurencyjne** - charakteryzują się pętlą zwrotną w warstwie ukrytej. Mogą być wykorzystane do generowania tekstu, tłumaczeń maszynowych, a także na przykład przewidywania cen rynkowych;
- **sieci samoorganizujące się** - wykorzystuje uczenie nienadzorowane. Składają się jedynie z warstwy wejściowej i wyjściowej. Zaś cechą charakterystyczną jest to, że neurony określające podobne klasy znajdują się obok siebie. Sieci te mogą być wykorzystywane do podziału klientów na odpowiednie grupy bądź do wskazania, jakim klientom zaproponować karty kredytowe [16, 17].

3.3. Głębokie uczenie

Głębokie sieci neuronowe (*ang. Deep Neural Network, DNN*) to podkategoria uczenia maszynowego. Koncentruje się na strukturyzacji uczenia maszynowego, gdzie komputery mogą rozpoznawać wzorce i podejmować decyzje jak ludzie. DNN to udoskonalenie podstawowych sieci neuronowych, więc część typów sieci opisanych w **sekcji 3.2** będzie odnosić się do DNN. Do takich sieci należy CNN. Głębokie sieci neuronowe to dynamicznie rozwijająca się dziedzina. Jest to możliwe dzięki zwiększającym się możliwościach

technicznych urządzeń, które udostępniają coraz więcej mocy obliczeniowej w coraz niższej cenie. Dodatkowy wpływ miały coraz większe zbiory danych, który koszt przechowywania stale maleje. Aby uzyskać odpowiednie wyniki z DNN należy wykorzystać ogromne zbiory danych, ponieważ jakoś rezultatu jest proporcjonalna do ilości dostarczonych danych.

Głębokie sieci neuronowe posiadają wiele warstw z milionami sztucznych neuronów połączonych ze sobą. Duża ilość warstw pozwala na zoptymalizowanie procesu rozwiązywania bardziej złożonych problemów. Sieci jednowarstwowe mogą tworzyć podstawowe prognozy. Sieci głębokie są zdolne do rozumienia złożonych wzorców oraz relacji przy jednoczesnym zwiększeniu dokładności predykcyjnej [18]. **Rysunek 3.8** przedstawia przykład sieci głębokiej. Taka sieć składa się z warstwy wejściowej, wielu warstw ukrytych i warstwy wyjściowej.



$i_x \forall x \in [1, 2, 3]$: dane wejściowe

$v_x \forall x \in [11, 12, 13, 21, 22, 23, 31, 32, 33]$: neurony w warstwie ukrytej

$o_x \forall x \in [1, 2]$: dane wyjściowe

Rys. 3.8. Schemat prostej głębokiej sieci neuronowej

Źródło: Opracowanie własne

Głęboka sieć neuronowa posiada bardzo dużo zastosowań, w których skład wchodzi m.in. generowanie treści, Deepfake, analiza obrazów, wskazywanie obiektów na obrazach, projektowanie leków, czatboty. Jest to udoskonalenie podstawowych sieci neuronowych [19].

4. Klasyfikacja danych

Człowiek od zawsze próbuje skategoryzować i uporządkować posiadaną wiedzę w zbiory pozwalające na łatwy dostęp do przechowywanych informacji na przykład w sposób tabelaryczny. Klasyfikacja to proces rozpoznawania obiektów na bazie analizy ich cech. Jest to jedna z pierwszych rzeczy, której uczą się niemowlęta: rozróżniania kształtów, kolorów czy własnych rodziców. W późniejszych latach te umiejętności są bardzo ważne, ponieważ w otaczającym świecie wiele rzeczy jest sklasyfikowanych. Przykładem takiej klasyfikacji są książki w katalogach bibliotecznych czy produkty spożywcze według jakości produkcji.

W uczeniu maszynowym klasyfikacja jest to metoda uczenia nadzorowanego, podczas której model próbuje przewidzieć etykietę obiektu na podstawie jego cech. Proces uczenia modelu klasyfikacji jest oparty o dwa zbiory, treningowy i testowy. Zbiór treningowy powinien być mniejszy od zbioru testowego. Po udanym procesie trenowania modelu następuje proces testowania, na podstawie którego wylicza się odpowiednie metryki pozwalające na ewaluację modelu. W pracy magisterskiej wykorzystano zbiór danych, który posiada dwa typy etykiet [0, 1], dlatego też na tej podstawie zbudowano macierz pomyłek.

4.1. Metryki klasyfikacji danych

W trakcie ewaluacji algorytmu służącego do klasyfikacji wykorzystuje się metryki klasyfikacji. Służą one do przewidywanie etykiet klas na podstawie danych wejściowych. W zależności od ilości klas wyjściowych rozróżniamy klasyfikację binarną i wieloklasową. W klasyfikacji binarnej występują tylko dwie klasy wyjściowe, a w klasyfikacji wieloklasowej już tych klas jest więcej. Przykładem klasyfikacji binarnej jest np. wykrywanie nieuchcianych wiadomości w skrzynce mailowej. Wiadomość przychodząca możemy określić jako 1 - „*pozytywną*”, bądź 0 - „*negatywną*”. Istnieje wiele sposobów na oceny klasyfikacji, do których zaliczamy m.in. dokładność, macierz pomyłek czy precyzję [20].

Macierz pomyłek pokazuje jakie wyniki możemy otrzymać w procesie klasyfikacji. Macierz ta została przedstawiona w **tabeli 4.1**. Jest to miara wydajności problemów klasyfikacji.

Tabela 4.1. Macierz pomyłek
Źródło: Opracowanie własne

		Prawdziwe wartości	
		1	0
Przewidziane wartości	1	TP	FN
	0	FP	TN

Wyróżniamy 4 rodzaje rezultatów:

- **TP** - prawdziwie pozytywny - przewidywano pozytywny wynik i otrzymano pozytywny wynik,
- **FN** - fałszywie negatywny - przewidywano negatywny wynik, a otrzymano pozytywny wynik,
- **FP** - fałszywie pozytywny - przewidywano pozytywny wynik, a otrzymano negatywny wynik,
- **TN** - prawdziwie negatywny - przewidywano negatywny wynik i otrzymano negatywny wynik.

Opierając się o macierz pomyłek można wyznaczyć następujące metryki:

- dokładność,
- precyza,
- czułość,
- F1
- swoistość
- ROC - AUC [20]

4.1.1. Dokładność

$$\frac{TP + TN}{TP + TN + FP + FN} \quad (4.1)$$

Dokładność określaczęstość poprawnego przewidywania klasyfikatora. Jest to stosunek wszystkich dobrze oznaczonych obiektów do liczby wszystkich prób [1, 20, 21].

4.1.2. Precyza

$$\frac{TP}{TP + FP} \quad (4.2)$$

Precyza wyjaśnia ile z prawidłowo przewidzianych przypadków jest poprawnych. Jest to stosunek poprawnie wybranych obiektów klasy „I”, do wszystkich wybranych obiektów tej klasy. Znajduje zastosowanie w przypadkach gdy wynik fałszywie dodatni jest większym problemem niż fałszywie ujemny [1, 20, 21].

4.1.3. Czułość

$$\frac{TP}{TP + FN} \quad (4.3)$$

Czułość określa ile rzeczywistych pozytywnych przypadków byliśmy w stanie poprawnie przewidzieć za pomocą naszego modelu. Jest to stosunek poprawnie sklasyfikowanych obiektów klasy „I”, do wszystkich obiektów, które powinny być w tej klasie. Czułość jest przydatną metryką w przypadkach gdy wynik fałszywie ujemny ma większe znaczenie niż wynik fałszywie dodatni [1, 20, 21].

4.1.4. Swoistość

$$\frac{TN}{TN + FP} \quad (4.4)$$

Swoistość lub specyficzność pozwala określić ile rzeczywiście uzyskało się przypadków negatywnych. Jest to stosunek poprawnie zaklasyfikowanych obiektów klasy „0”, do wszystkich obiektów, które powinny być w tej klasie [20].

4.1.5. F1

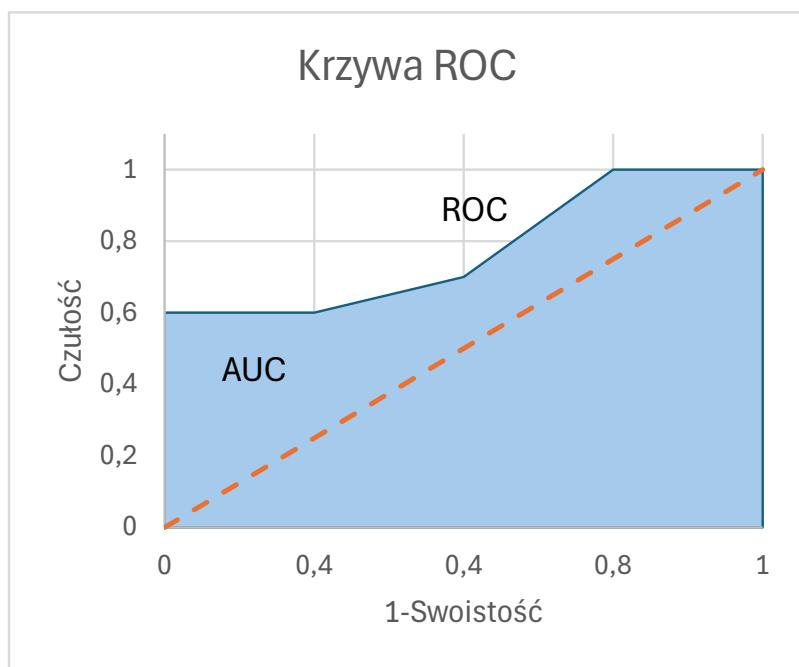
$$\frac{2 * x * y}{x + y} \quad (4.5)$$

Parametr F1 pozwala na połączoną ocenę precyzji i czułości. Jest to średnia harmoniczna precyzji (x) i czułości (y). Parametr F1 może być skuteczną metryką oceny w następujących przypadkach:

- wyniki fałszywie pozytywne i fałszywie negatywne są równie kosztowne
- zwiększenie ilości danych nie zmienia skuteczności wyniku
- wynik prawdziwie ujemny jest wysoki [20].

4.1.6. ROC - AUC

Krzywa ROC-AUC przedstawiona na **rysunku 4.1**, pozwala na ocenę sprawności klasyfikatora. ROC jest krzywą prawdopodobieństwa, którą wykreśla czułość w stosunku do swoistości. Wykres przedstawia wydajność (dokładność predykcji) modelu klasyfikacji. AUC (*ang. Area Under Roc Curve*) jest to pole pod krzywą ROC. AUC to miara zdolności klasyfikatora do rozróżniania klas.



Rys. 4.1. Przykład krzywej ROC
Źródło: Opracowanie własne

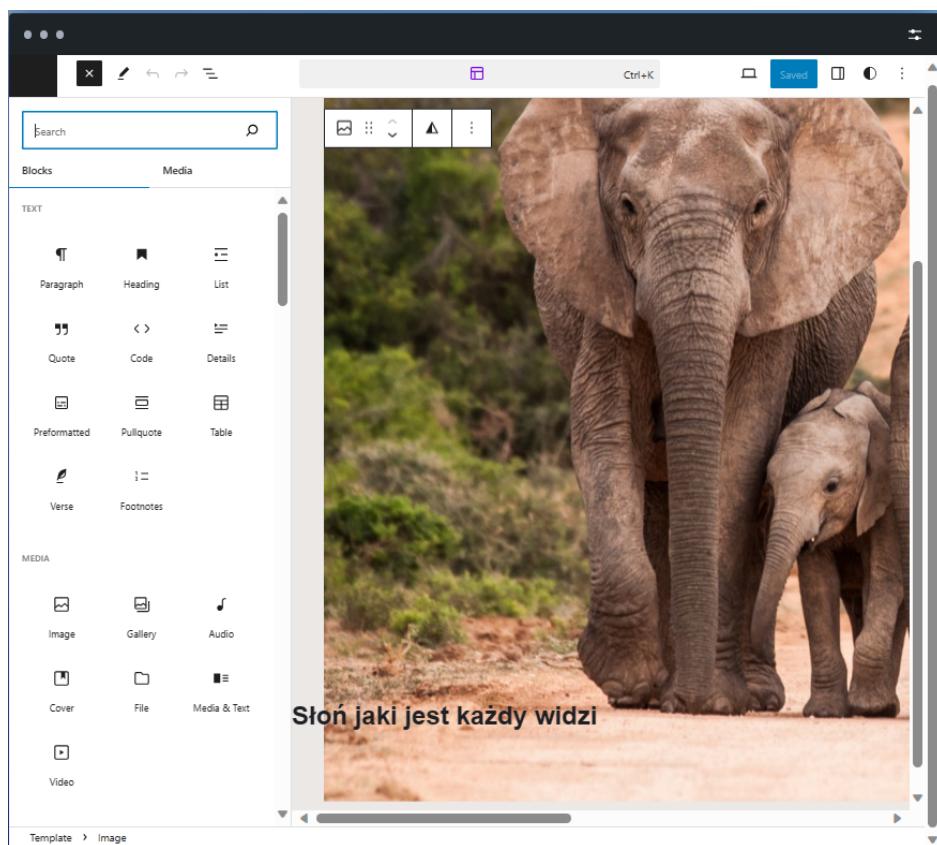
Im większa jest wartość AUC tym lepsza wydajność modelu w rozróżnianiu klas pozytywnych i negatywnych. Wynik AUC jest z zakresu $[0, 1]$:

- $AUC = 1$ - klasyfikator idealny, który rozróżnia wszystkie punkty,
- $0,5 \leq AUC < 1$ - klasyfikator prawie idealny, czyli z dużym prawdopodobieństwem sklasyfikuje poprawnie obiekty
- $AUC = 0,5$ - klasyfikator losowy, oznacza to, że nie ma pewności czy klasyfikator jest w stanie rozpoznać cechy, czy może przypisuje je losowo,
- $AUC < 0,5$ - klasyfikator gorszy niż klasyfikator losowy [20, 22].

5. Platformy low-code/no-code

Low-Code oraz no-code to nowe podejście skupiające się na tworzeniu oprogramowania bez konieczności znajomości języka programowania. Celem tych działań jest umożliwienie osobom nietechnicznym, które nie są programistami, na łatwe i szybkie tworzenie aplikacji biznesowych. Pozwoli to odpowiedzieć na rosnące zapotrzebowanie na wytwarzanie oprogramowania, rozwiązań biznesowych, przy brakującej liczbie programistów, która mogłaby sprostać potrzebom rynku.

Podejście low-code/no-code jest dość nowym rozwiązaniem. Początki tego podejścia można zauważać w narzędziach umożliwiających samodzielne tworzenie stron internetowych np *Wordpress*, *Joomla* czy *Wix*. Dzięki tym rozwiązaniom, stronę można utworzyć w łatwy sposób, za pomocą tzw. kafelków. Kafelki to komponenty, które pozwalają na umieszczenie tekstu, grafiki, wideo czy innych elementów na stronie [23–25]. Na **rysunkach 5.2, 5.1** przedstawiono przykłady rozbudowanych komponentów. Można zauważać, że możliwe jest rozszerzone formatowanie tekstu czy wstawianie elementów multimedialnych. Użytkownik nie musi znać języka programowania czy sposobów oznaczania za pomocą języka znaczników HTML, tylko dodaje wybrany przez siebie kafel.

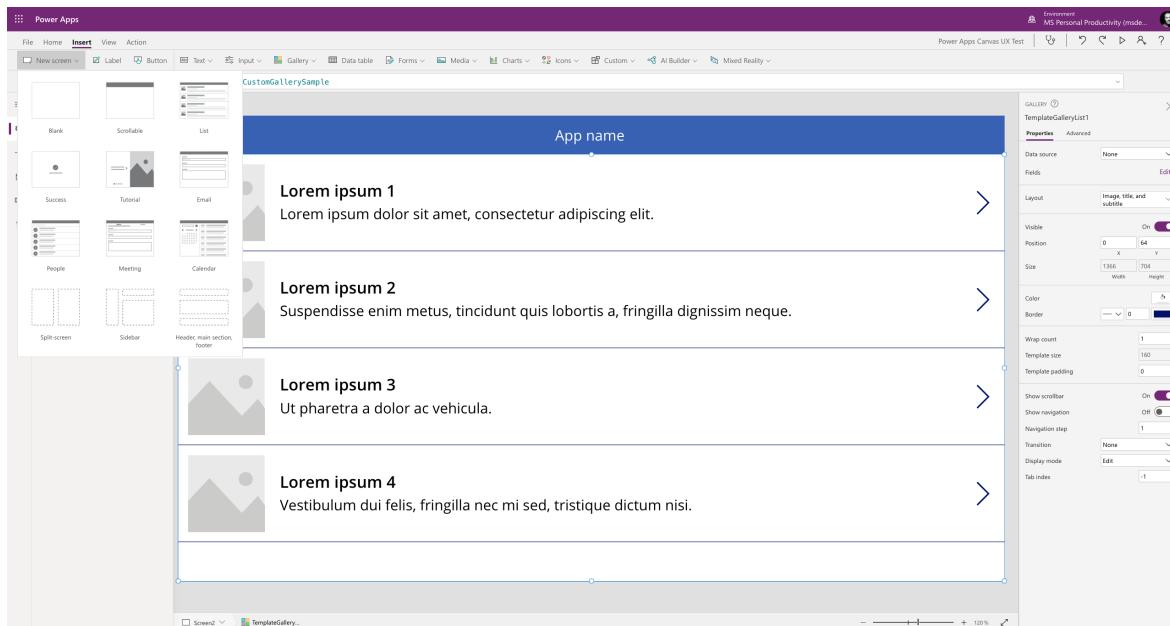


Rys. 5.1. Wordpress.com
Źródło: [26]

Rozwój narzędzi do tworzenia stron oraz chęć automatyzacji coraz większej liczby obszarów programistycznych, rozwinął dziedzinę low - code. Wśród użytkowników coraz większą popularnością cieszą się platformy low-code (*ang. Low-code Development Platforms, LCDPs*). Takie rozwiązania dostarczają m.in. Google, Microsoft czy Amazon. Pozwalają one na tworzenie wysoko skalowalnych rozwiązań przy minimalnym lub zerowym nakładzie pracy z kodem. Umożliwia to osobom z niewielkim doświadczeniem w programowaniu, na szybkie wdrożenie oraz tworzenie niezawodnego oprogramowania. Twórcy platform oferują również korzystającym, zmniejszenie ilości pracy potrzebnej do wdrożenia albo rozwijania kolejnych funkcjonalności. LCDP udostępniane twórcom aplikacji, umożliwiają skalowalność rozwiązań tworzonych na własne potrzeby. Dodatkowo są popularne przy tworzeniu aplikacji typu „*aplikacja jako usługa*” (*ang. Software-as-a-Service, SaaS*), które opłacane są tylko za stopień ich użycia. To podejście w konkretnych sytuacjach może się okazać dużo bardziej opłacalne niż utrzymywanie swoich rozwiązań serwerowych. Dzięki takim rozwiązaniom wiele małych firm będzie mogło pozwolić sobie na tworzenie i utrzymywanie dostosowanych rozwiązań opartych o ekosystem *Microsoft365 / Google Workspace* [27, 28].

5.1. Microsoft PowerApps

Jest to platforma programistyczna umożliwiająca tworzenie niestandardowych aplikacji dla rozwiązań biznesowych. Przykład ekranu platformy przedstawiono na **rysunku 5.2.**



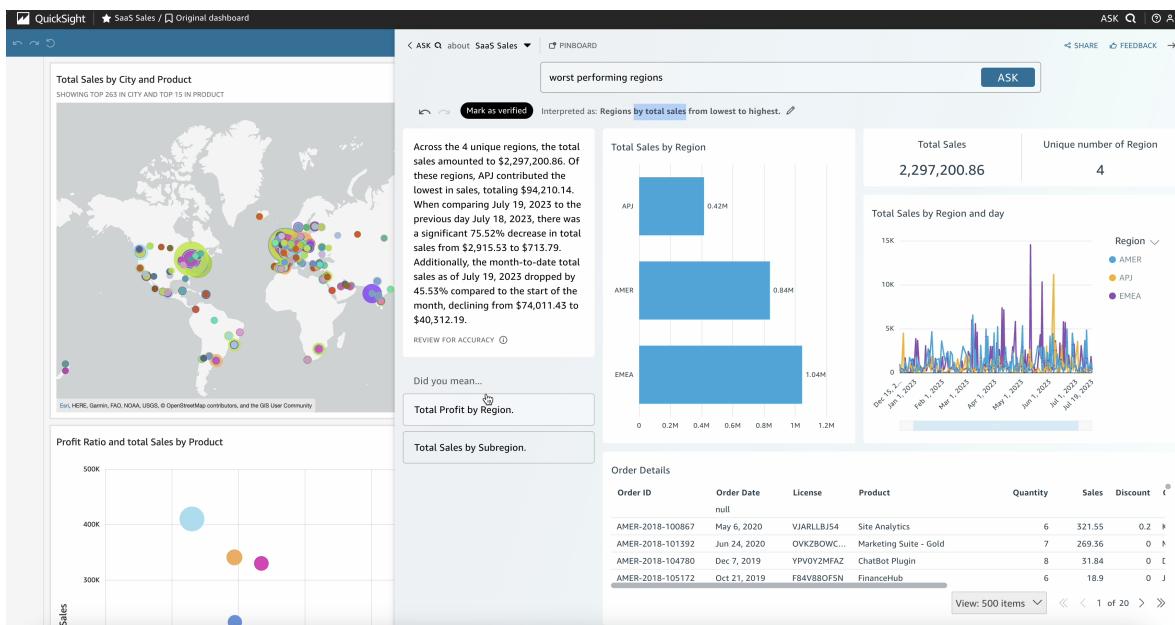
Rys. 5.2. PowerApps od Microsoft
 Źródło: [29]

Microsoft PowerApp umożliwia tworzenie aplikacji opartych o różnorodne źródła danych. Do nich należą między innymi: SQL Server, SharePoint, Dynamics 365. Zaletą MS PowerApp jest tworzenie aplikacji responsywnych, działających dobrze na wielu rodzajach urządzeń. Pozwala też tworzyć trzy typy aplikacji przy braku konieczności kodowania

- **Kanwa** - typ aplikacji oparty o model danych znajdujący się np. w Excelu. Kanwę tworzy się za pomocą przesuwanych kafelek, użytkownik układa je samodzielnie. Umożliwia to pełną dowolność w tworzonym interfejsie graficznym.
- **Oparte na modelu** - aplikacja wygenerowana przy użyciu Microsoft Dataverse. Bazuje na danym modelu danych oraz pozwala na jego analizę. Użytkownik może analizować dane, bez znajomości specjalistycznych komend np. z języka Python.
- **Karty** - aplikacje, które można dodać do usługi Microsoft Teams w określonym biznesowym celu. Rozwiążanie to wprowadza możliwość tworzenia aplikacji zespołowo, bez konieczności przełączania się między różnymi platformami. Zaletą tego rozwiązania jest możliwość tworzenia małych aplikacji do pojedynczej funkcjonalności [30–33].

5.2. Amazon QuickSight

Jest to rozwiązanie firmy Amazon, które umożliwia firmom dostarczanie rozwiązań z zakresu analityki biznesowej (*ang. business intelligence, BI*). Amazon QuickSight udostępnia interaktywne pulpity (*ang. dashboards*), zawierające narzędzia do analizy danych. Użytkownicy mają możliwość korzystania z interaktywnych formularzy, raportów czy zapytań w języku naturalnym. Rozwiążanie korzysta z jednego źródła prawdy, określonego przez użytkownika [34]. Przykładowy *dashboard* przedstawia rysunek 5.3.

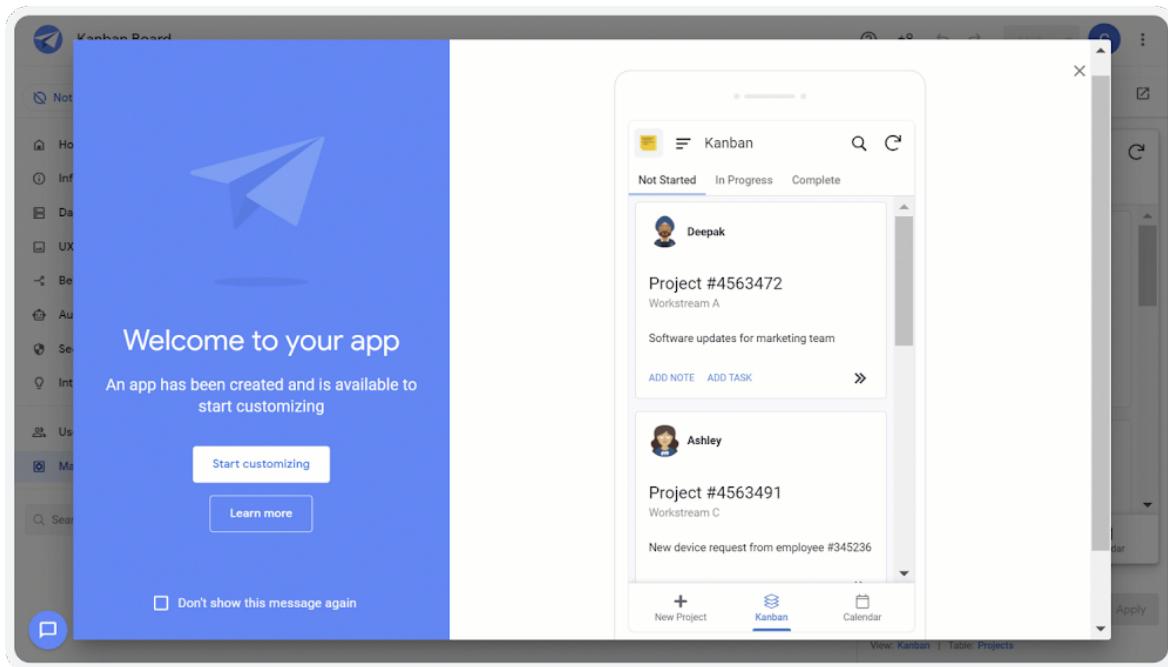


Rys. 5.3. Amazon QuickSight

Źródło: [34]

5.3. Google AppSheet

Platforma AppSheet od firmy Google umożliwia tworzenie aplikacji mobilnych oraz desktopowych bez użycia kodu. Firma wskazuje na możliwości integracyjne z różnymi dostawcami danych, do których należą między innymi Microsoft, Dropbox. Dodatkowo posiada wbudowaną integrację z aplikacjami Google Workspace, do których należą Gmail, Sheets oraz Spaces. Platforma pozwala również na tworzenie automatycznych botów, które wykonują zadania po interakcji z bodźce zewnętrznymi bądź wewnętrznymi. Narzędzie pozwala w prosty sposób na tworzenie szybkich rozwiązań biznesowych w ekosystemie firmy Google [35]. Przykładowy ekran tworzenia aplikacji prezentuje **rysunek 5.4**



Rys. 5.4. Google AppSheet
Źródło: [35]

6. Microsoft Azure

Jest to rozwiązanie chmurowe udostępnione przez firmę Microsoft. Zawiera wiele usług i narzędzi z zakresu zarządzania IT. W trakcie rejestracji na platformie nie jest pobierana opłata. Opłacie podlega jedynie wykorzystywanie konkretnych usług oraz zasobów platformy (*ang: pay-per-use*).

Początki platformy sięgają 2008 roku, kiedy do użytku został oddany Windows Azure. Narzędzie było zbudowane z modułów Windows NT. Windows Azure zawierał ograniczoną liczbę usług w chmurze. W kolejnych latach rozbudowano Windows Azure o relacyjne bazy danych SQL Azure oraz obsługę języków takich jak Java czy PHP. W 2010 oddano po raz pierwszy platformę do użytku komercyjnego. W tej wersji dodano m.in.

- .NET Framework 4 związanego z Microsoft SQL Server,
- obsługę aplikacji pisanych w wielu językach (takich jak C#, Java, PHP)
- sieć dostarczania zawartości (*ang. Content Delivery Network, CDN*).

Następnie wdrożono oprogramowanie typu open source oraz infrastrukturę definiowaną jako serwis (*ang. Infrastructure-as-a-Service, IaaS*). Zmieniono też nazwę na Microsoft Azure. W kolejnej generacji Microsoft zaadaptował rozwiązania Big Data do swojej platformy, umożliwiając korzystanie z języka R, połączenie do Power BI, a także umożliwienie połączenia do rozwiązań end-to-end.

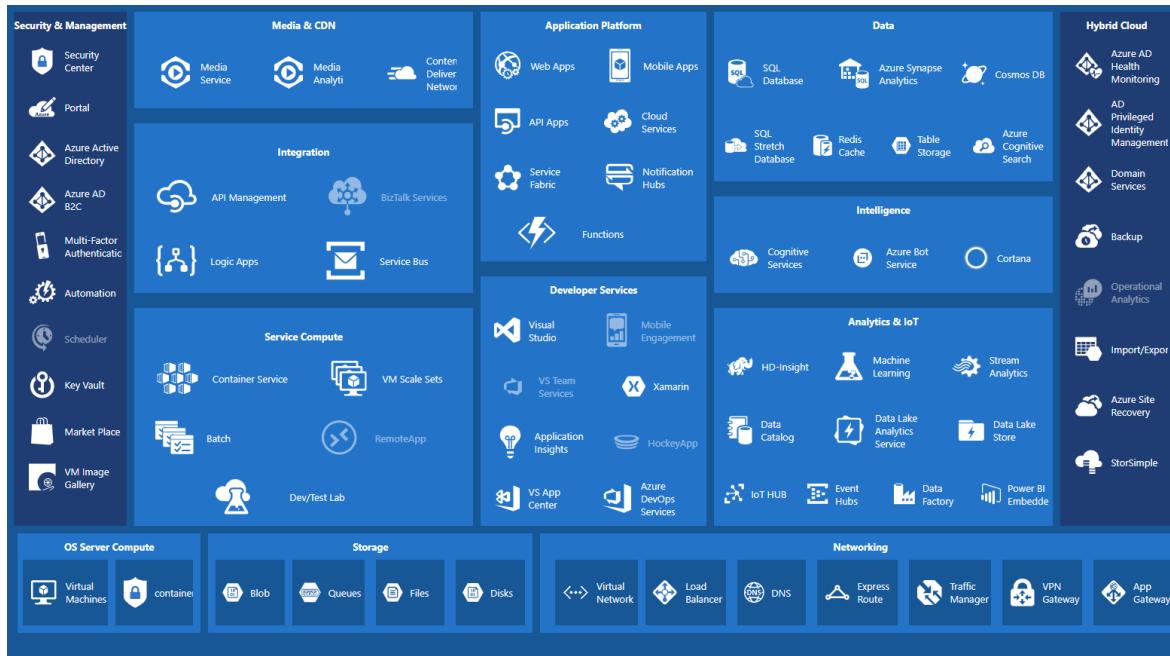
W czwartej generacji platformy, Microsoft skupił się na rozwiązaniach uczenia maszynowego oraz integracji z bazami danych, dzięki czemu powstało Azure Machine Learning Studio oraz Azure Machine Learning Operations (MLOps).

Obecnie platforma została wzbogacona o Kuberntesa, dzięki czemu konteneryzacja ułatwia pracę z klastrami wirtualnymi. Wirtualne klastry pozwalają na wydajniejszy i wygodniejszy sposób zarządzania aplikacjami i usługami. Dodatkowo zostało udostępnione wiele kombinacji usług technologicznych:

- aplikacja jako usługa (*ang. Software-as-a-Service, SaaS*),
- interfejs jako usługa (*ang. Infrastrucuture-as-a-Service, IaaS*),
- platforma jako usługa (*ang. Platfrom-as-a-Service, PaaS*).

Dzięki powyższym usprawnieniom Microsoft dostarcza platformę przyjazną użytkownikowi, która umożliwia korzystanie z ponad 200 różnych usług opartych m.in. o rozwiązania chmurowe i sztuczną inteligencję [36–38].

Rysunek 6.1 pokazuje obecny podział usług na platformie Microsoft Azure. Można wyróżnić wiele obszarów m.in. z zakresu bezpieczeństwa, zarządzania danymi, usług deweloperskich, analizy danych czy platformy aplikacji.

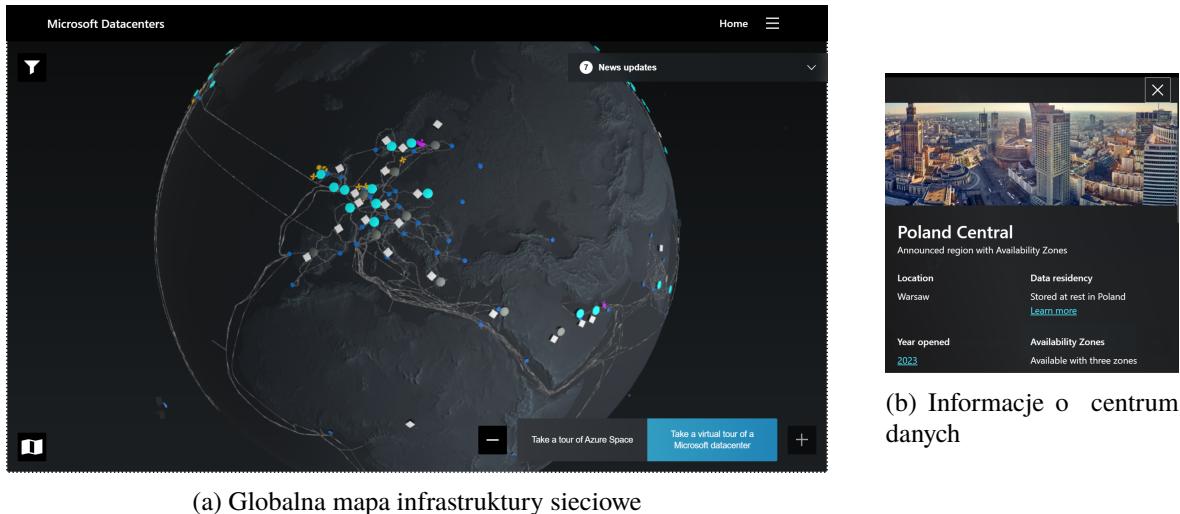


Rys. 6.1. Schemat podziału usług MS Azure
Źródło: [39]

6.1. Infrastruktura

Infrastruktura globalna Azure składa się z dwóch części: fizycznej infrastruktury oraz globalnej łączności. Infrastruktura fizyczna składa się z ponad 200 centrów danych rozmieszczonych na całym świecie, połączonych w jedną globalną sieć. Takie rozwiązanie umożliwia wysoką skalowalność i dostępność poszczególnych rozwiązań. Cały ruch sieciowy jest utrzymywany w prywatnej sieci Microsoft. Pozwala to na zachowanie informacji o adresach IP wewnętrznych sieci, a co za tym idzie, informacje te nie trafiają do opinii publicznej [40].

Na swojej stronie internetowej Microsoft udostępnia wirtualną mapę, umożliwiającą zobaczenie aktualnej sieci Microsoftu oraz jej rozmieszczenie na globie ziemskim. Podgląd przedstawiono na **rysunku 6.2**.



(a) Globalna mapa infrastruktury sieciowej

(b) Informacje o centrum danych

Rys. 6.2. Zdjęcia z mapy insfrastruktury Microsoft Azure
Źródło: [41, 42]

Interaktywna mapa prezentuje informację o poszczególnych krajach oraz centrach danych znajdujących się na terytoriach tych krajów.

6.2. Machine Learning Studio

Azure Machine Learning (ML) to rozwiązanie dostarczane przez Microsoft Azure. Opiera się o ideę low-code/no-code, która została opisana w **rozdziale 5**. Usługa ML umożliwia łatwe i szybkie tworzenie wysoce wydajnych modeli uczenia maszynowego, a także zarządzanie nimi. Użytkownik nie musi posiadać wiedzy teoretycznej związanej z uczeniem maszynowym czy programowaniem. Rozwiązanie wspiera pełen cykl życia kompleksowego uczenia maszynowego. Taki cykl obejmuje następujące etapy:

- etykietowanie i przygotowywanie danych,
- budowanie i trenowanie modeli SI i ML,
- walidację i wdrażanie procesu uczenia się,
- zarządzanie i monitorowanie uzyskanych wyników [43].

Platforma umożliwia tworzenie potoków zadań, które połączone w jeden potok, wykonują poszczególne zadania w odpowiedniej kolejności. Mechanizm „*złap i upuść*” (ang. *drag&drop*) umożliwia tworzenie potoków w sposób graficzny. Dzięki modułowej budowie potoków można uzyskać rozwiązanie wielokrotnego użytku. W ramach jednego doświadczenia każdy moduł może być buforowany. Pozwala to na „zapamiętanie” wyniku poprzedniego uruchomienia i wykorzystanie go ponownie (o ile dane wejściowe albo konfiguracja nie uległy zmianie). Dodatkowo poza predefiniowanymi operacjami można wykorzystać moduły języka Python/R. Kolejną możliwością jest wytworzenie rozwiązania w technologii „**Jupyter Notebook**” oraz wizualne narzędzie wykorzystujące mechanizm *przeciągnij i upuść* (ang. *Drag & Drop*). Rozwiązanie to pozwala układać „*kafelki*” służące do tworzenia potoków zadań. Każde zadanie wykorzystuje wcześniej przygotowaną jednostkę obliczeniową, dzięki czemu można przewidzieć albo dostosować koszt wykorzystania modelu. Umożliwione zostało również wdrażanie modeli jako punktów końcowych. Pozwala to na komunikowanie się z nimi za pomocą REST API.

Microsoft pobiera płatność jedynie za użytkowanie usług, co oznacza, że jeśli kластer komputerowy był wykorzystywany jedynie przez 1 godzinę, to za tą jedną godzinę zostanie obciążony klient [44].

7. Doświadczenie

W ramach niniejszej pracy dyplomowej przeprowadzono doświadczenie polegające na porównaniu autorskiego algorytmu klasyfikacji (stworzonego w ramach pracy inżynierskiej pt. „Wykorzystanie algorytmów genetycznych w systemach wykrywania intruzów w sieciach komputerowych” [1]) wraz z:

- algorytmami klasyfikacji danych dwuklasowych dostępnymi w środowisku Microsoft Azure,
- algorymem DANet [45, 46]

Algorytmy opisano w kolejnych podrozdziałach. Doświadczenie przeprowadzono w następujących etapach:

1. Określenie założeń technicznych
2. Wybór danych i algorytmów klasyfikacji
3. Konfiguracja programistycznego środowiska badawczego
4. Utworzenie doświadczenia
5. Przeprowadzenie badań
6. Analiza wyników.

7.1. Metodologia badawcza

W celu oceny jakości algorytmów klasyfikacji określono następujący protokół metodologii badawczej, który został przedstawiony w **tabeli 7.1**. Na początku zdefiniowano problem badawczy, określono do niego pytania badawcze oraz postawiono przewidywane hipotezy.

Tabela 7.1. Metodologia badawcza
 Źródło: Opracowanie własne

<p>Problem badawczy: Czy algorytm klasyfikacji danych utworzony w ramach pracy inżynierskiej może konkurować z rozwiązaniami dostępnymi w środowiskach komercyjnych</p>
<p>Pytania badawcze:</p> <ol style="list-style-type: none">1. Czy algorytm jest konkurencyjny pod względem wybranych metryk:<ul style="list-style-type: none">– dokładność algorytmu– czas działania– precyzja– czułość– F1– auc

Hipotezy:

1. Nie ma istotnej różnicy pomiędzy wynikami próby testowej i treningowej nr. 1.
2. Nie ma istotnej różnicy pomiędzy wynikami prób testowych.
3. Wynik dopasowania algorytmów w *Próbie 1* nie przekracza dolnej granicy przedziału ufności dla próby testowej

7.2. Założenie techniczne

Dane prezentowane w **Tabeli 7.2** określają podstawowe założenia techniczne przyjęte w trakcie wykonywania analizy porównawczej. Dane te dotyczą między innymi środowiska, w którym wykonane było doświadczenie. Dodatkowo uwzględniono zestaw danych oraz biblioteki użyte w trakcie tworzenia doświadczenia.

Tabela 7.2. Założenia techniczne pracy dyplomowej
 Źródło: Opracowanie własne

Środowisko uruchomieniowe	Machine Learning Studio[47]
Język programowania	Python 3.x
Wykorzystane biblioteki	scikit-learn [6] Numpy [48] Pandas [49, 50]
Wykorzystane dane	CICDS2017 [51]

7.3. Dane

Zbiór danych został przygotowany przez Kanadyjski Instytut Cyberbezpieczeństwa działający przy Uniwersytecie Nowy Brunszwik. Został wykonany za pomocą narzędzia CICFlowMeter [52]. Zbiór zawiera 79 cech ruchu sieciowego, do których zaliczyć można:

1. etykietę,
2. czas trwania przesyłu,
3. minimalną długość pakietu zwrotnego,
4. maksymalną długość pakietu zwrotnego,
5. port docelowy,
6. długość pakietów.

Zbiór pozwala na określenie czy ruch sieciowy jest życzliwy (*ang. BENING*), czy nieżyczliwy (różne możliwe formy ataku na sieć). Dodatkowo dane zostały podzielone na pięć dni roboczych: poniedziałek 3.07.2017 - piątek 7.07.2017. Dane z poniedziałku zawierają jedynie ruch życzliwy. W pozostałe dni zostały zasymulowane ataki na sieć komputerową [1, 53].

7.4. Algorytmy wykorzystane w doświadczeniu

W trakcie eksperymentu zastosowano różne algorytmy klasyfikacji danych, by móc ocenić jakość autorskiego rozwiązania Gausian Naive Bayes - with GA. Do porównania wykorzystano następujące algorytmy takie jak:

1. Two-Class Support Vector Machine
2. Two-Class Boosted Decision Tree
3. Two-Class Decision Forest
4. Two-class Neural Network
5. Two-Class Average Perceptron
6. DANet.

Algorytmy 1 - 5 są dostępne na platformie Microsoft Azure, natomiast algorytm 6 to rozwiązanie udostępnione przez jednego z autorów pracy na platformie Github [45].

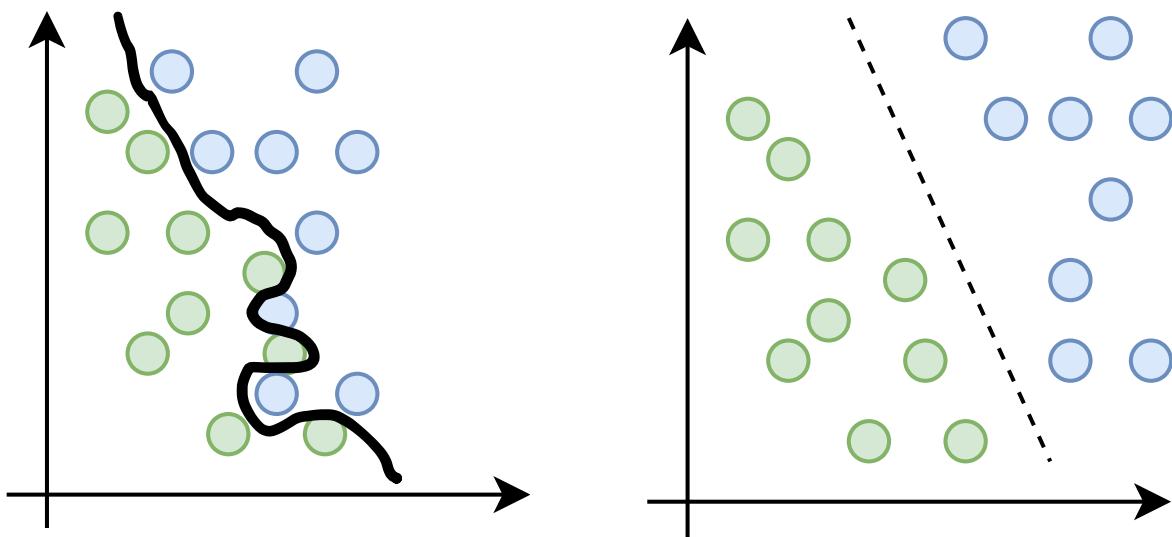
Charakterystyczną cechą tych algorytmów jest klasyfikacja ukierunkowana na 2 kategorie wejściowe. W tym wypadku są to kategorie ruchu sieciowego:

- **BENIGN** (*pl. życzliwy*),
- **OTHER** (*pl. inne*), gdzie inne to pozostałe typy ruchu sieciowego.

W kolejnych podrozdziałach pokrótko scharakteryzowano zastosowane rozwiązania.

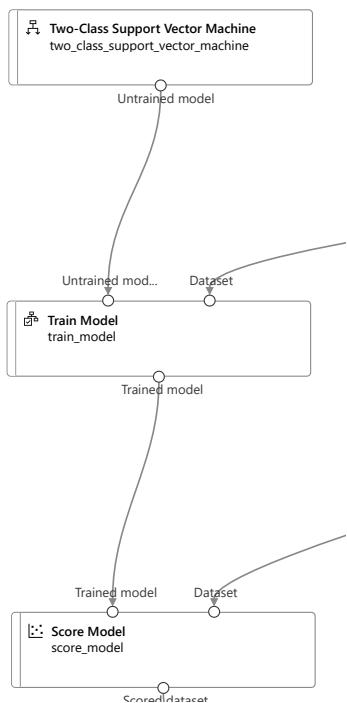
7.4.1. Two-Class Support Vector Machine

Algorytm SVM ma za zadanie znaleźć hiperpłaszczyznę w przestrzeni K-wymiarowej (gdzie K - liczba cech), która rozdziela zbiory punktów odpowiadające różnym klasom. W pierwszej kolejności SVM szuka separatora między klasami, a następnie przekształca się dane w taki sposób, by móc przekształcić separator w hiperpłaszczyznę [54]. Sposób działania został zobrazowany za pomocą **wykresów 7.1**.



Rys. 7.1. Schemat SVM
Źródło: [55]

Implementacja algorytmu w Azure Machine Learning została przedstawiona na **rysunku 7.2.**

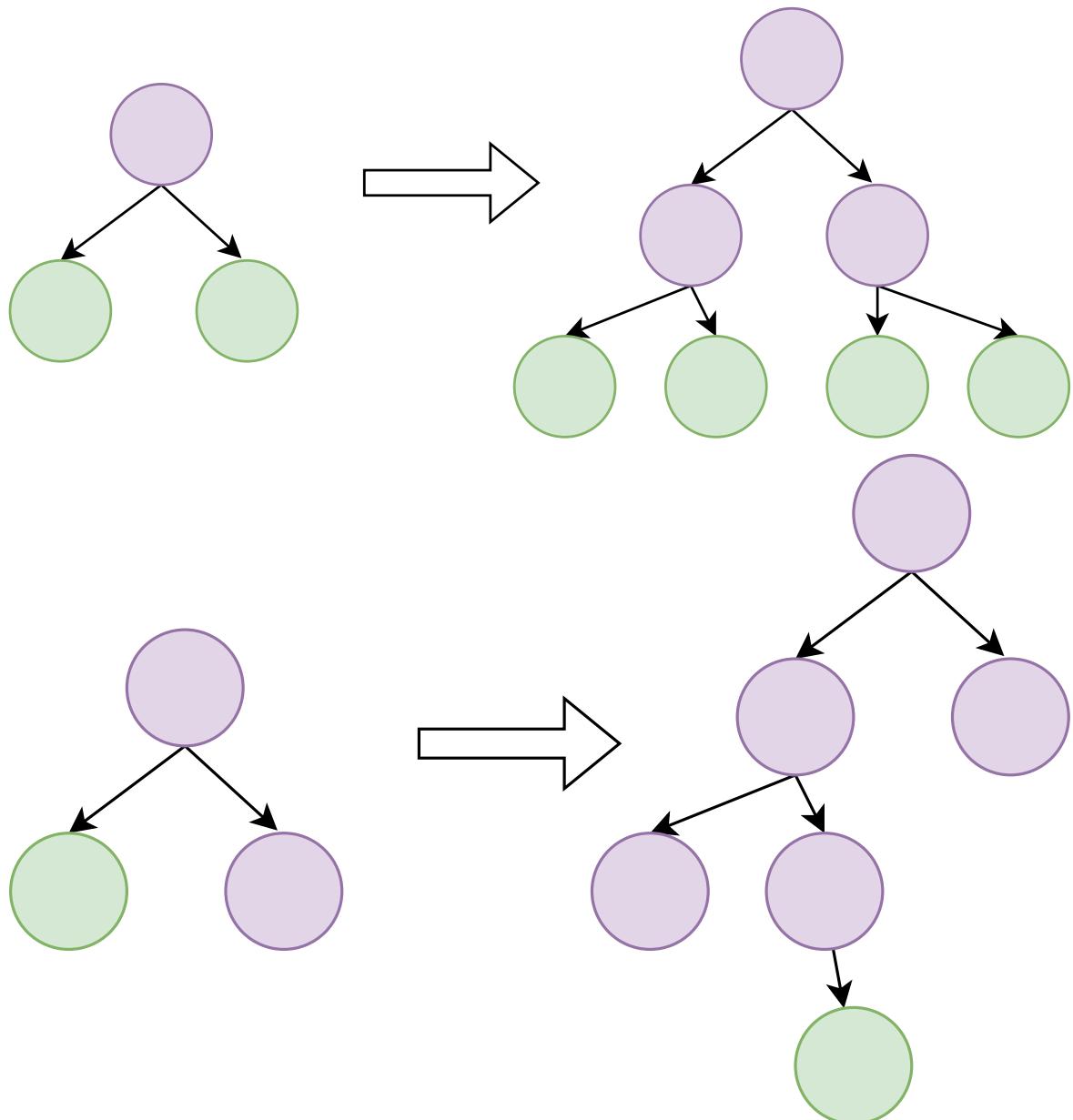


Rys. 7.2. Potok zadań dla modelu *Two-Class Support Vector Machine*
Źródło: Opracowanie własne

Jest to część potoku, która obrazuje schemat działania procesu trenowania i sprawdzania algorytmu. Pierwszy kafelek obrazuje model SVM, który jest połączony do elementu odpowiadającego za trening modelu. Do tego samego miejsca są połączone dane treningowe. Po wykonaniu zadania trenowania następuje przejście do zadania oceniającego model. Do tego zadania połączone są dane testowe. Przy pomocy danych testowych następuje ewaluacja modelu. Wyniki ewaluacji są przekazywane do zbiorczej tabeli.

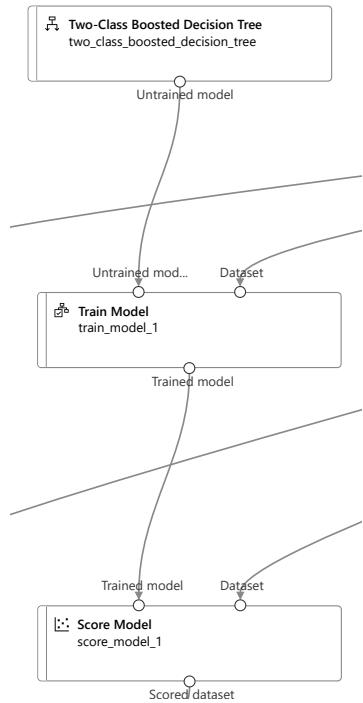
7.4.2. Two-Class Boosted Decision Tree

Jest to algorytm drzewa decyzyjnego oparty o algorytm LightGBM. Dzięki zastosowaniu takiego podejścia algorytm działa szybciej oraz ma mniejszą złożoność obliczeniową. Algorytm ten działa na zasadzie doboru odpowiedniego liścia, zamiast jak w przypadku klasycznych algorytmów opartych na drzewie, wyboru odpowiedniej warstwy [56]. Sposób podejścia liściastego został ukazany na **schemacie 7.3**.



Rys. 7.3. Sposób działania algorytmu
Źródło: [56]

Model wykorzystywany w Azure ML został ukazany na **rysunku 7.4**.

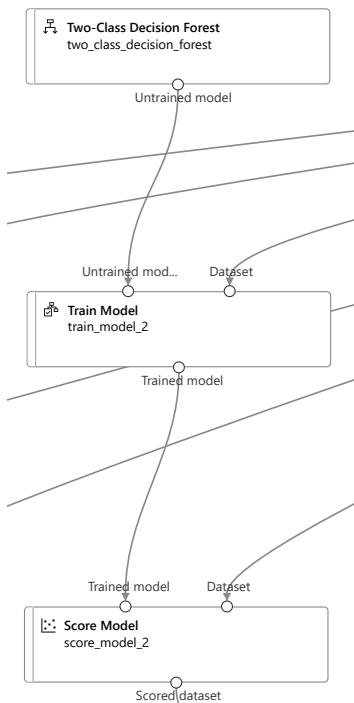


Rys. 7.4. Potok zadań dla modelu *Two-Class Boosted Decision Tree*
 Źródło: Opracowanie własne

Potok jest zbudowany analogicznie do potoku opisanego w **podsekcji 7.4.1**. Do bloku treningowego został podłączony model wzmacnionego drzewa decyzyjnego.

7.4.3. Two-Class Decision Forest

Las decyzyjny to algorytm, którego wynik opiera się o agregację wyników wielu drzew decyzyjnych. Uzyskanie wyniku zależy od algorytmu trenowania lasu. Przykładowo w klasifikacji losowym lasem wieloklasowym (*ang. Multi-class random forest classification*), każde drzewo głosuje na jedną klasę. Klasa, która zostanie wybrana większością głosów, zostaje uznana za wynikową [57]. Model wykorzystany w Azure ML pokazano na **modelu 7.5**.

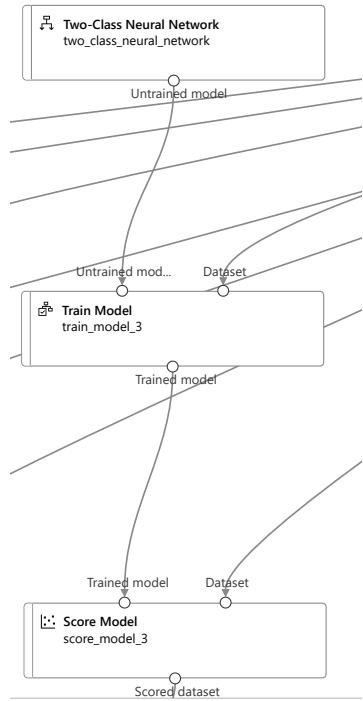


Rys. 7.5. Potok zadań dla modelu *Two-Class Decision Forest*
 Źródło: Opracowanie własne

Potok został zbudowany analogicznie do potoku opisanego w **podsekcji 7.4.1**. Do bloku treningowego został podłączony model lasu decyzyjnego.

7.4.4. Two-class Neural Network

Jest to sieć neuronowa, która składa się z warstwy wejściowej, trzech warstw ukrytych (każda posiada po 100 węzłów), oraz z warstwy wyjściowej. Przykładowa sieć neuronowa została zobrazowana na **schemacie 3.6**. Moduł wykorzystany w Azure ML ukazano na **rysunku 7.6**.

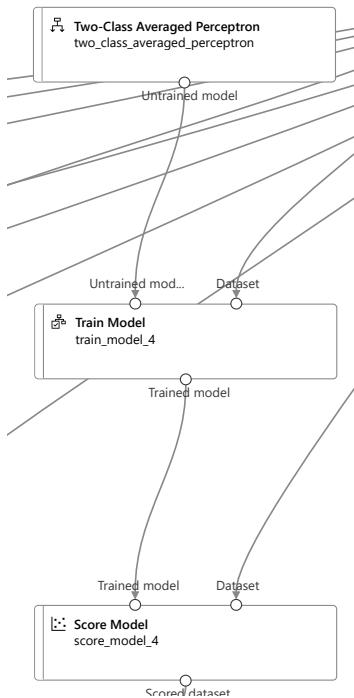


Rys. 7.6. Potok zadań dla modelu *Two-Class Neural Network*
 Źródło: Opracowanie własne

Potok jest zbudowany analogicznie do potoku opisanego w **podsekcji 7.4.1**. Do bloku treningowego został podłączony model sieci neuronowej.

7.4.5. Two-Class Average Perceptron

Jest to najprostsza odmiana sieci neuronowej, czyli pojedynczy perceptron, który jest matematycznym modelem neuronu. Składa się on z n wejść, takiej samej ilości wag, progu Θ , sumatora, funkcji aktywującej i wyjścia. Został zobrazowany na **schemacie 3.7**. Może służyć za prosty klasyfikator binarny albo za regresor. Model wykorzystany w Azure ML ukazano na **zdjęciu 7.7**.

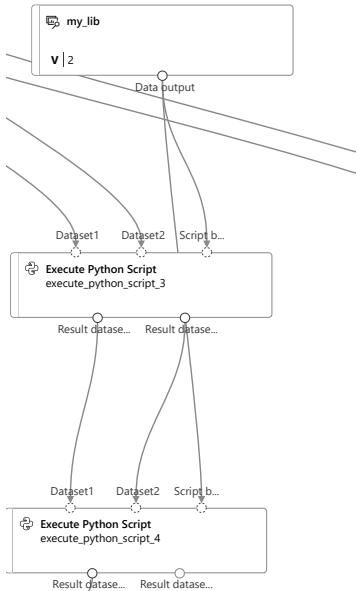


Rys. 7.7. Potok zadań dla modelu *Two-Class Average Perceptron*
 Źródło: Opracowanie własne

Potok jest zbudowany analogicznie do potoku opisanego w **podsekcji 7.4.1**. Do bloku treningowego został podłączony model klasycznego perceptronu.

7.4.6. Gausian Naive Bayes - with GA

Algorytm ten polega na połączeniu algorytmu genetycznego (GA) wraz z klasyfikatorem naiwnym Bayesa wykorzystującego rozkład Gaussa (GNB). Zadaniem algorytmu genetycznego jest znalezienie najistotniejszych cech w zbiorze tabelarycznym. Poszukiwane cechy powinny pozwolić na zmniejszenie wymiarowości danych oraz na zmniejszenie kosztów obsługi samego klasyfikatora. Co może zostać uzyskane późniejszych etapach testowania, ze względu na zmniejszoną ilość danych wymaganych do przetworzenia. GA wykorzystywał w metodzie **fitness** algorytm GNB w celu określenia dopasowania danych. Zadaniem GNB było znalezienie najlepszej dostępnej kombinacji cech, które pozwalały na uzyskanie najlepszego dopasowania [1]. Model wykorzystywany w Azure ML różni się od gotowych modeli tym, że dołączono do niego bibliotekę napisaną w języku Python, która zawiera kod wykorzystywany w pracy inżynierskiej autora [58].



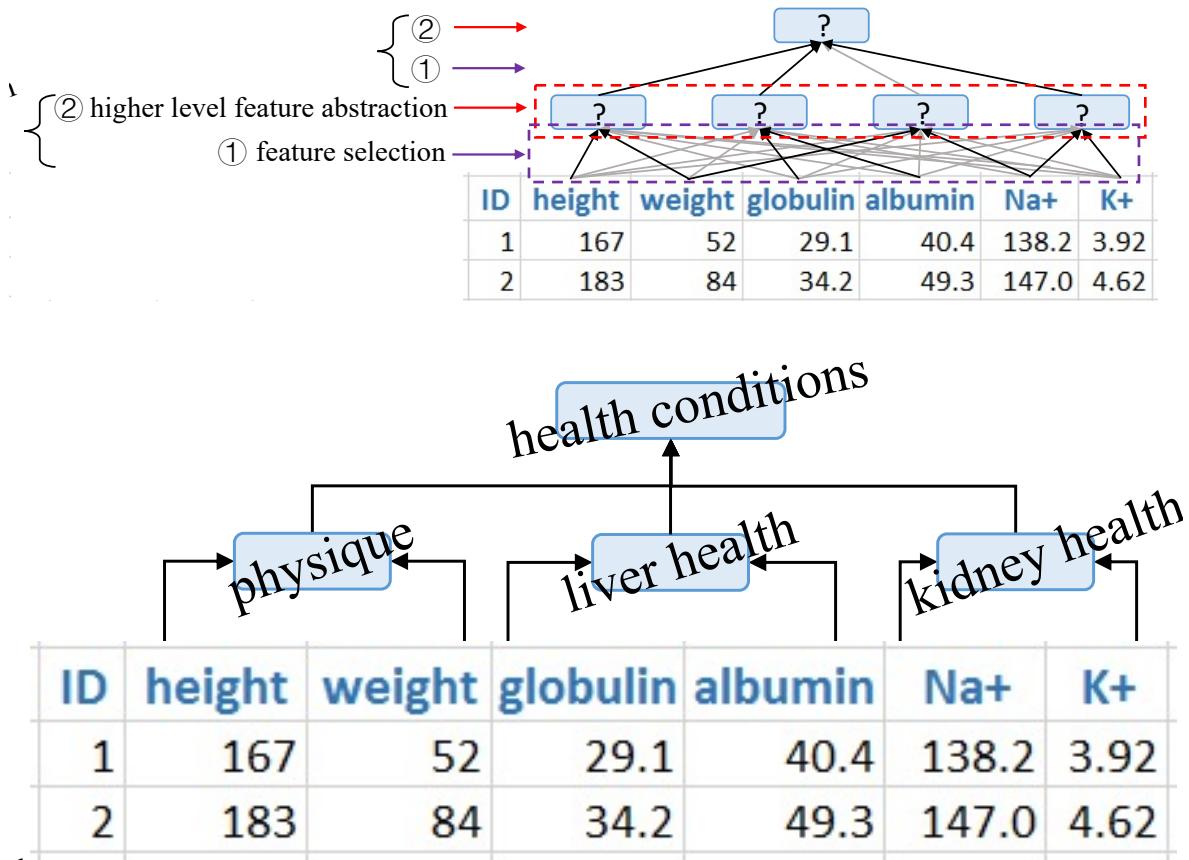
Rys. 7.8. Potok zadań dla modelu
 Źródło: Opracowanie własne

Potok został stworzony przy wykorzystaniu 3 głównych elementów:

- **my.lib** - jest to blok danych zawierający archiwum .zip, w którym znajduje się biblioteka odpowiadająca za algorytm wykorzystywany w doświadczeniu,
- **Execute Python Script (execute_python_script_3)** blok wykonuje operację trenowania znajdującej się w bibliotece dołączonej do bloku. Do tego celu potrzeba było podłączyć dane testujące i treningowe do pierwszego bloku, by przekazać je dalej wraz z wytrenowanym modelem.
- **Execute Python Script (execute_python_script_4)** zadanie wykonuje operację testowania modelu, która znajduje się również w dołączonej bibliotece. Wynikiem obliczeń jest tabela z obliczonymi metrykami, która zostaje przekazana i dołączona do tabeli zbiorczej.

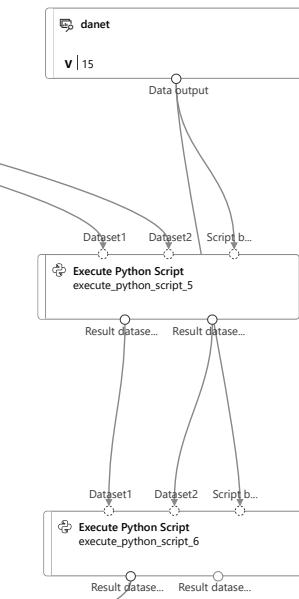
7.4.7. DANet

Twórcy tego algorytmu wprowadzają dodatkową warstwę abstrakcyjną o nazwie „*Abstract Layer*”. Warstwy te budują sieć o nazwie „*Deep Abstract Network*” (DANet). Zadanie dodatkowych warstw jest grupowanie cech w skorelowanych zbiorach. Zbiory te budują sieć powiązań między sobą w formie sieci semantycznej. Gdy sieć semantyczna jest zbudowana, to w ostatnim kroku wykonywana jest klasyfikacja w trzywarstwowej sieci perceptronów (ang. *Multilayer Perceptron network, MLP*) [45, 46]. Sposób działania przedstawiono na **schemacie 7.9**.



Rys. 7.9. Sposób działania DANet
Źródło: [46]

Model znajdujący się z Azure ML został przedstawiony na **zdjęciu 7.10.**

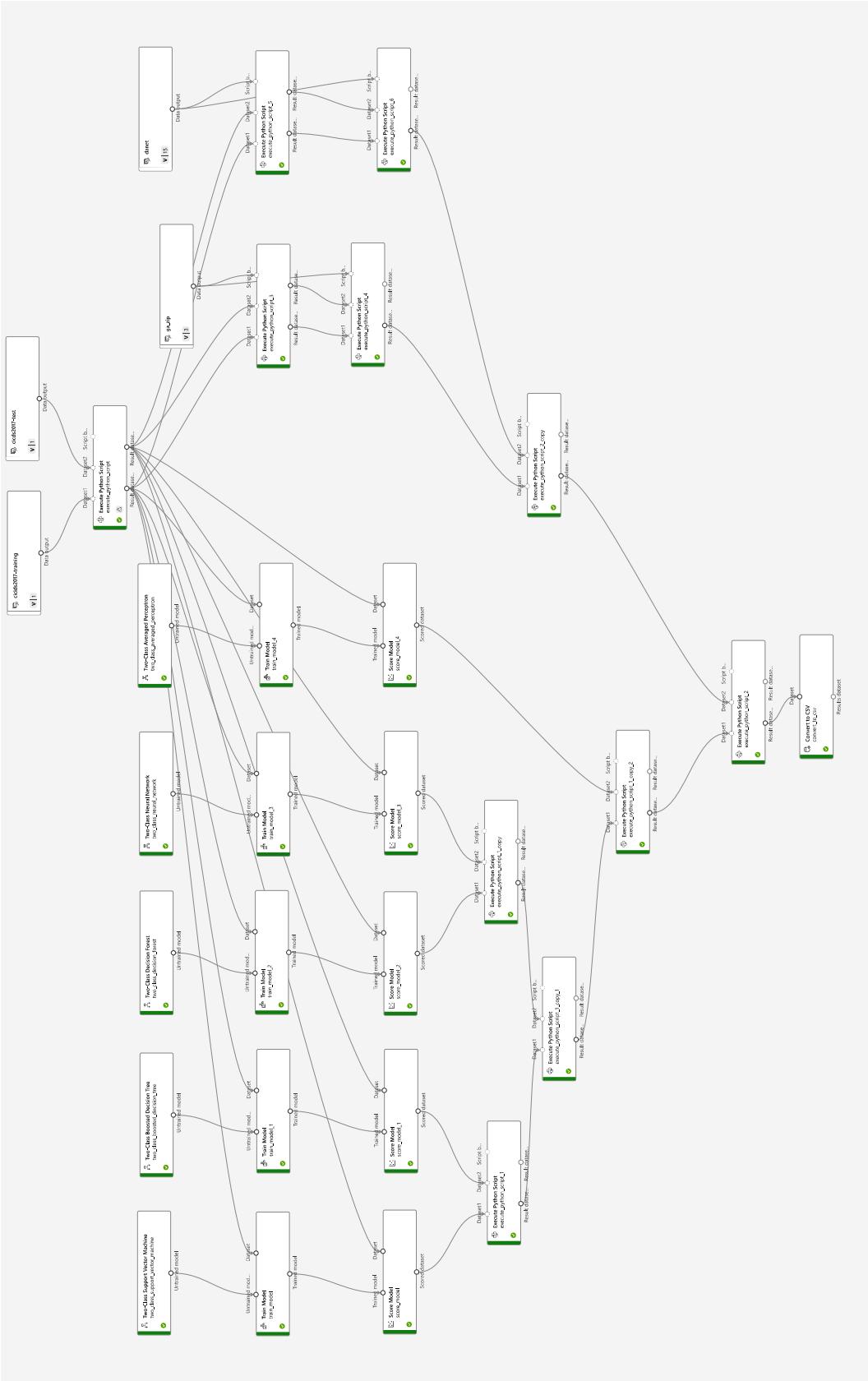


Rys. 7.10. Potok zadań dla modelu *DANet*
Źródło: Opracowanie własne

7.5. Programistyczne środowisko badawcze

Analizę porównawczą algorytmów wykonano na platformie Microsoft Azure. Wykorzystano usługę Azure Machine Learning Studio. Wybrano to rozwiązanie, ponieważ umożliwia uniezależnienie obliczeń od komputera lokalnego. Dodatkowo, w łatwy sposób pozwala tworzyć skomplikowane potoki zadań, składające się z komponentów wielokrotnego użytku. Każdy komponent uruchamia się w środowisku odizolowanym od pozostałych operacji. Dzieje się tak dzięki wykorzystaniu wielowęzłowych klastrów obliczeniowych, bazujących na oprogramowaniu Docker. Klastry te mogą skalować się w zależności od potrzeb oraz dostępnej jednostki [59].

Całe doświadczenie zostało odwzorowane w graficznym potoku narzędzia „*Projektant*” oraz przedstawione na **zdjęciu 7.11.**



Rys. 7.11. Potok zadań
Źródło: Opracowanie własne

7.6. Przebieg eksperymentu

„Projektant”, dostępny w Azure Machine Learning umożliwił utworzenie interaktywnego potoku zadań. Potok ten składa się z kilku części:

- Przygotowanie i obróbka zbiorów danych
- Trenowanie oraz testowanie algorytmów klasyfikacji danych
- Utworzenie tabeli porównawczej dla wyników poszczególnych algorytmów (**obraz 7.11**).

Poszczególne kroki przebiegu eksperymentu zostały opisane na kolejnych stronach.

7.6.1. Przygotowanie danych

W pierwszym kroku do normalizacji danych wykorzystano język Python oraz biblioteki Pandas i Numpy. Za pomocą wymienionych narzędzi, etykiety słowne zostają zamienione na wartości **0** i **1** oraz następuje zamiana wartości $[NaN, -inf, inf]$ na liczbę 0. Cały proces został zobrazowany na **diagramie 7.12**



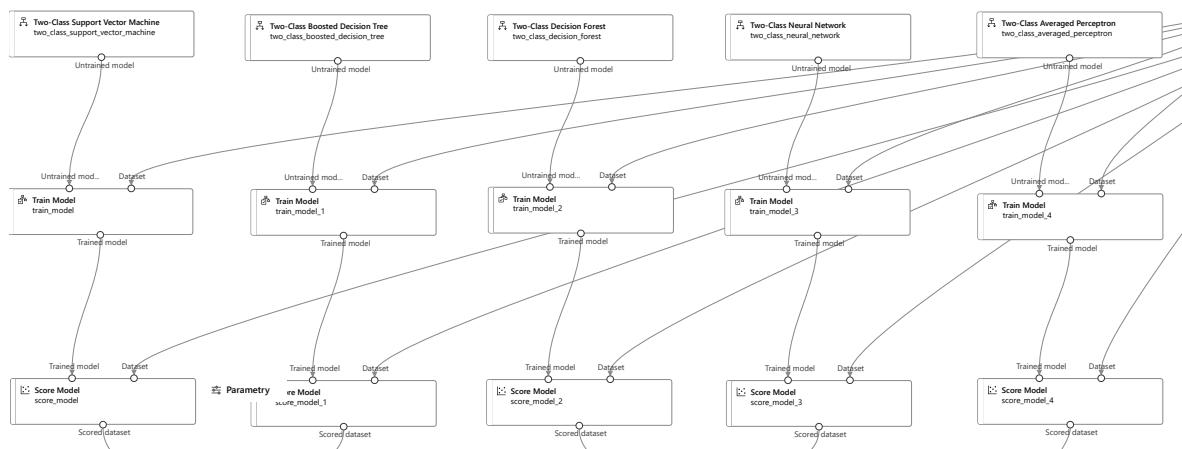
Rys. 7.12. Potok normalizacji danych
Źródło: Opracowanie własne

7.6.2. Trenowanie oraz testowanie algorytmów

Kolejną grupą zadań widoczną w potoku są te związane z trenowaniem i testowaniem poszczególnych algorytmów opisanych w **rozdziale 7**. Każdy test składa się 3 kafelek. W przypadku algorytmów dostarczonych wraz z platformą Azure ML są to:

- **model klasyfikujący** - odpowiada za przygotowanie algorytmu klasyfikacyjnego
- **blok treningowy** - tworzy wytrenowany model, za pomocą połączonego zbioru danych
- **blok ewaluacyjny** - sprawdza wcześniej wytrenowany model za pomocą powiązanego zbioru danych.

Potok zadań wykorzystujący algorytmy dostarczone przez Microsoft Azure został ukazany na **schemacie 7.13**

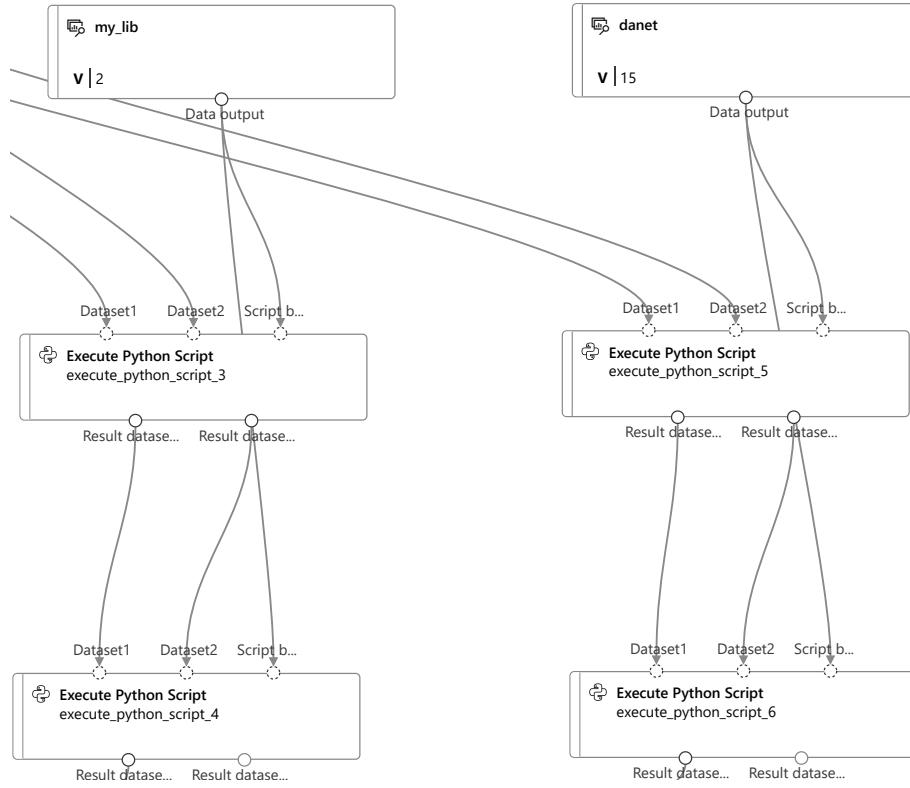


Rys. 7.13. Potok zadań dla algorytmów klasyfikacyjnych
Źródło: Opracowanie własne

Algorytmy dostarczone w ramach pracy badawczej składają się z:

- **biblioteka Python** - archiwum o rozszerzeniu **.zip**, które zawiera w sobie odpowiednie pliki napisane w języku Python
- **blok treningowy** - wykorzystuje dostarczoną bibliotekę do wytrenowania modelu oraz zapisania na platformie Azure najlepszego uzyskanego wyniku za pomocą powiązanego zbioru danych
- **blok ewaluacyjny** - wykorzystuje dostarczoną bibliotekę do ewaluacji algorytmu za pomocą połączonego zbioru danych

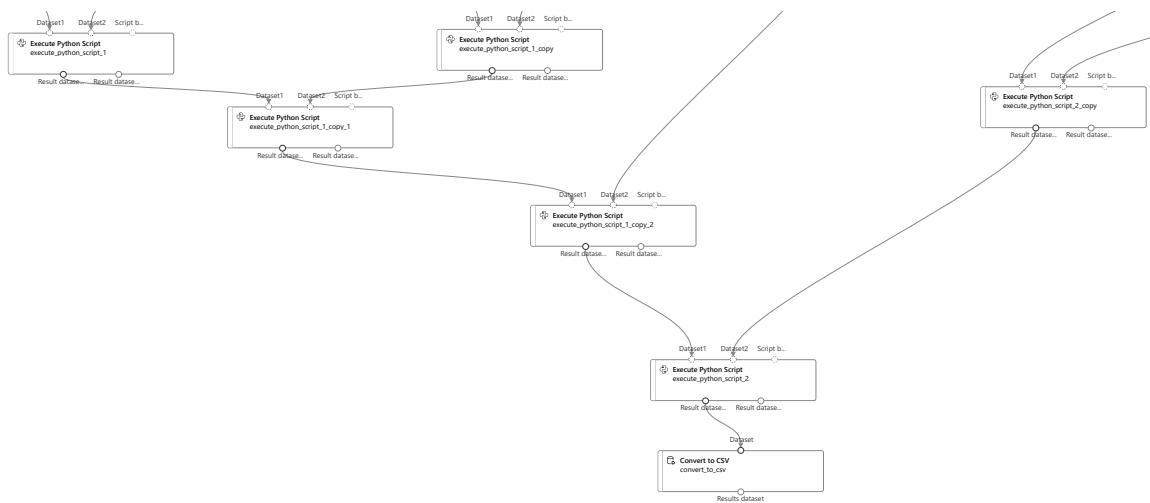
Potok zadań dla algorytmów niestandardowych został ukazany na **rysunku 7.14**



Rys. 7.14. Potok zadań dla algorytmów klasyfikacyjnych
Źródło: Opracowanie własne

7.6.3. Utworzenie tabeli porównawczej

Kolejną częścią zadań jest zebranie wyników poszczególnych algorytmów oraz połączenie ich w jedną całość. Wykorzystano do tego moduły języka Python, które zwracają przetworzone wyniki oraz łączą je w jedną tabelę zbiorczą, co pokazano na **rysunku 7.15**.



Rys. 7.15. Moduły odpowiedzialne za przetworzenie wyników
Źródło: Opracowanie własne

7.6.4. Wyniki danych treningowych

Aby zweryfikować działanie całego procesu klasyfikacji wykorzystano znormalizowane dane treningowe do wytrenowania oraz przetestowania działania algorytmów klasyfikacyjnych. Cały proces trwał „**1 dzień 10 godzin 55 minut 53 sekundy**”. Wyniki tych działań widać na **rysunku 7.16**. Wykres przedstawia wyniki dla metryk klasyfikacyjnych jak:

- dokładność
- precyzja
- czułość
- F1
- AUC.

Analizując wykres można zauważyć, że uzyskane wyniki znajdują się w przedziale [94%, 100%] w każdej metryce co pokazuje jakość każdego z algorytmów, a także to, że algorytmy poradziły sobie niemal bezbłędnie w rozpoznawaniu ruchu sieciowego, na którym były uczone. Zbiór, który wykorzystano do trenowania oraz testowania danych zawierał w sobie 225805 wpisów z czego 97718 należało do klasy „1”, zaś 128027 należało do klasy „0”. Liczba klas wyjściowych wynosi 2, przez wzgląd na to, że jest to klasyfikacja binarna, która klasyfikuje dane jako *1* albo *0*.

Tabela 7.3. Liczba elementów należących do danej klasy w zbiorze treningowym

Źródło: Opracowanie własne

Klasa	Liczba wystąpień
1	97718
0	128027
Suma	225805

Bazując na tym zbiorze oraz uzyskanych wynikach udało się udowodnić poprawność działania procesu klasyfikacji wieloma algorytmami genetycznymi.



Rys. 7.16. Wyniki testów algorytmów klasyfikacyjnych na danych treningowych
 Źródło: Opracowanie własne

7.7. Wyniki danych testowych

Aby uzyskać realne wyniki podczas porównywania poszczególnych algorytmów zastosowano zbiór treningowy opisany w **tabeli 7.3** oraz zbiór testowy, który zawierał 2273097 wpisów należących do klasy „1” oraz 557646 wpisów należących do klasy „0”. Sumarycznie ilość wpisów wynosi: 2830743, co zostało pokazane w **tabeli 7.4**. Pomiary testowe powtórzone 2 razy dzięki czemu uzyskano 3 próby badawcze.

Tabela 7.4. Liczba elementów należących do danej klasy w zbiorze testowym

Źródło: Opracowanie własne

Klasa	Liczba wystąpień
1	2273097
0	557646
Suma	2830743

W analizie porównawczej algorytmów zastosowano następujące metryki klasyfikacji jak:

- dopasowanie
- precyza
- czułość
- F1
- AUC

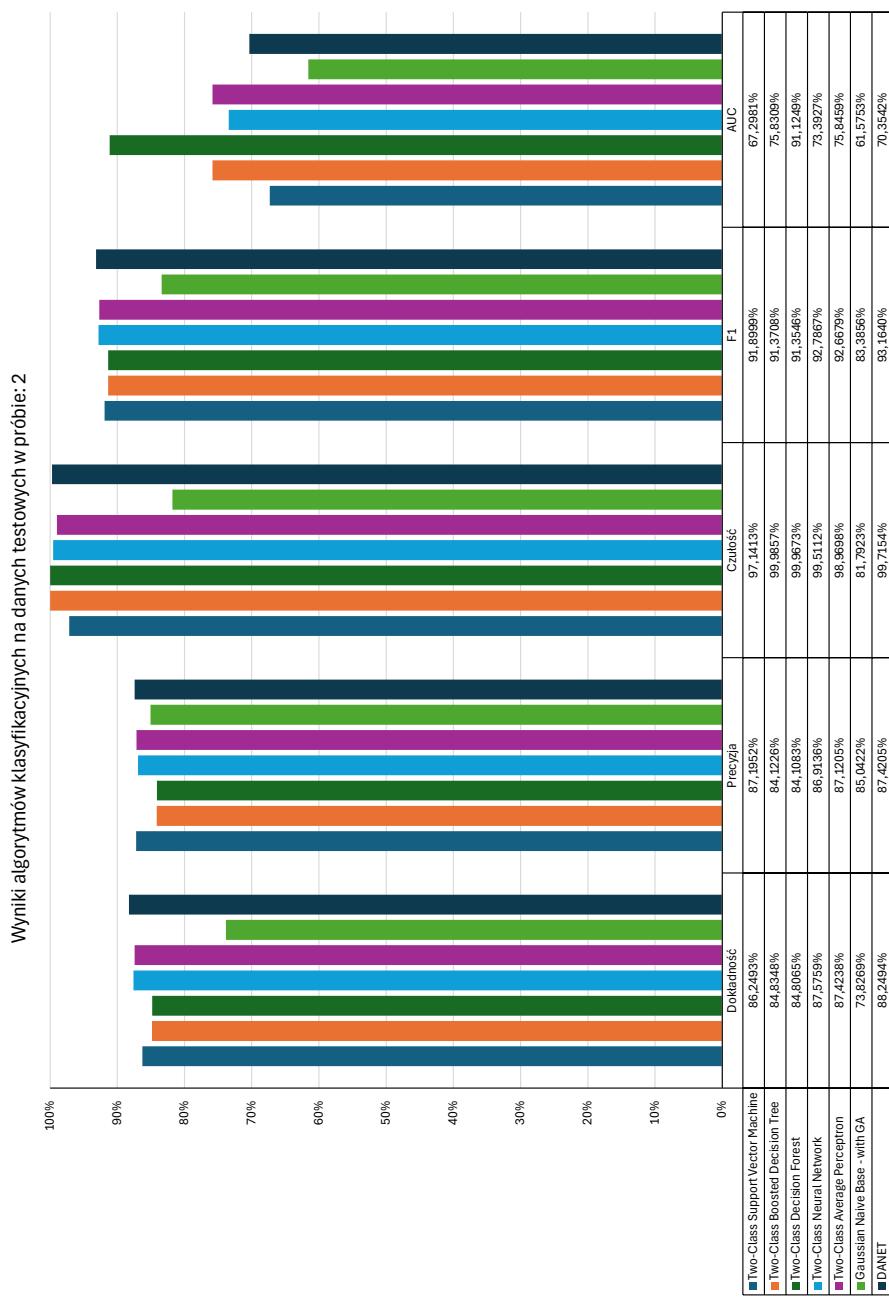
Poniżej na **rysunkach: 7.17, 7.18, 7.19** zostały przedstawione wyniki zbiorcze dla poszczególnych metryk. Dodatkowo przedstawiono również wynik pomiaru treningowego, który w większości przypadków jest wyższy od danych testowych. Co prawdopodobnie jest spowodowane różnicą w ilości danych testowych i treningowych. Dodatkowo w każdej kolumnie oznaczono kolorem zielonym najwyższy wynik dla danej metryki, a kolorem czerwonym najniższy wynik dla danej metryki.

Wyniki algorytmów klasyfikacyjnych na danych testowych w próbie: 1



Rys. 7.17. Wyniki testów algorytmów klasyfikacyjnych na danych testowych dla próby 1

Źródło: Opracowanie własne



Rys. 7.18. Wyniki testów algorytmów klasyfikacyjnych na danych testowych dla próby 2

Źródło: Opracowanie własne



Rys. 7.19. Wyniki testów algorytmów klasyfikacyjnych na danych testowych dla próby 3
 Źródło: Opracowanie własne

7.8. Analiza uzyskanych wyników

Aby uzyskać realne wyniki podczas porównywania poszczególnych algorytmów zastosowano zbiór treningowy opisany w **tabeli 7.3** oraz zbiór testowy opisany w **tabeli 7.4**.

7.8.1. Wyniki dopasowania

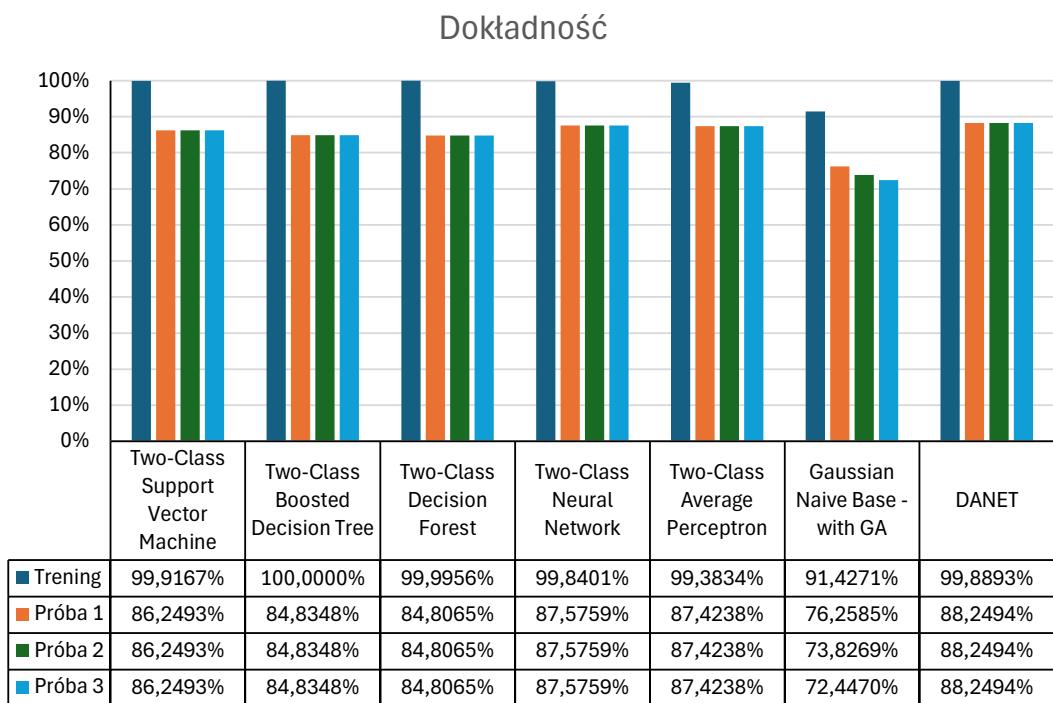
Najlepszy wynik dopasowania dla danych testowych uzyskał algorytm *DANet*, który poprawnie rozpoznał 88,2494% próbek. Najgorszy wynik uzyskał algorytm *GAGNB* z dopasowaniem w zbiorze [72,4470%; 73,8269%; 76,2585%]. Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* z wynikiem 100,00%, a najgorszy *GAGNB* z wynikiem 91,4271%. Wyniki dopasowania dla poszczególnych prób zostały przedstawione na **tabeli 7.5** oraz na **wykresie 7.20**.

Tabela 7.5. Wynik dopasowania algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik dopasowania			
	Próba Treningowa	Próba 1	Próba 2	Próba 3
Two-Class Support Vector Machine	99,9167%	86,2493%	86,2493%	86,2493%
Two-Class Boosted Decision Tree	100,0000%	84,8348%	84,8348%	84,8348%
Two-Class Decision Forest	99,9956%	84,8065%	84,8065%	84,8065%
Two-Class Neural Network	99,8401%	87,5759%	87,5759%	87,5759%
Two-Class Average Perceptron	99,3834%	87,4238%	87,4238%	87,4238%
Gaussian Naive Base - with GA	91,4271%	76,2585%	73,8269%	72,4470%
DANET	99,8893%	88,2494%	88,2494%	88,2494%



Rys. 7.20. Dokładność algorytmów

Źródło: Opracowanie własne

7.8.2. Wyniki precyzji

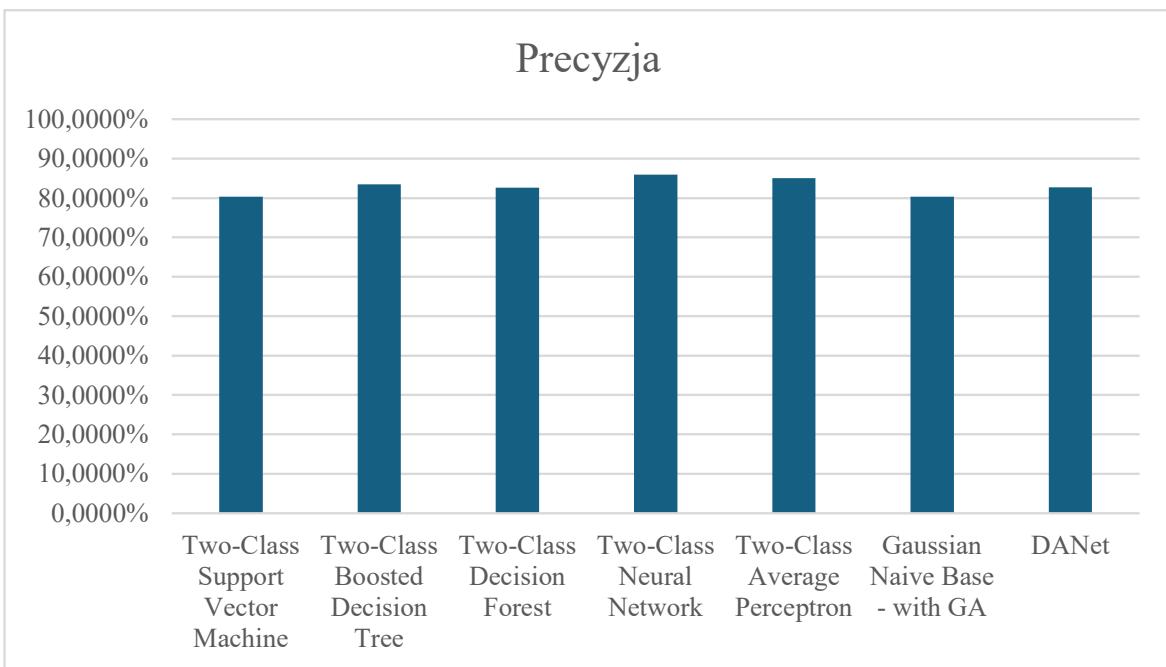
Najlepszy wynik precyzji dla danych testowych uzyskał algorytm *DANet* (87,4205%). Najgorszy wynik uzyskał algorytm *Two-Class Decision Forest* (84,1084%). Algorytm *GAGNB* uzyskał wyniki: ([85,0422%;, 85,4461%; 86,2876]). Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* (100,00%), a najgorszy *GAGNB* (94,2285%). Wyniki precyzji dla poszczególnych prób zostały przedstawione w tabeli 7.6 oraz na wykresie wykresie 7.21.

Tabela 7.6. Wynik precyzji algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik precyzji			
	Próba Treningowa	Próba 1	Próba 2	Próba 3
Two-Class Support Vector Machine	99,8742%	87,1952%	87,1952%	87,1952%
Two-Class Boosted Decision Tree	100,0000%	84,1226%	84,1226%	84,1226%
Two-Class Decision Forest	99,9898%	84,1083%	84,1083%	84,1083%
Two-Class Neural Network	99,9231%	86,9136%	86,9136%	86,9136%
Two-Class Average Perceptron	99,8510%	87,1205%	87,1205%	87,1205%
Gaussian Naive Base - with GA	94,2285%	85,4461%	85,0422%	86,2876%
DANET	99,8568%	87,4205%	87,4205%	87,4205%



Rys. 7.21. Precyza algorytmów
Źródło: Opracowanie własne

7.8.3. Wyniki czułości

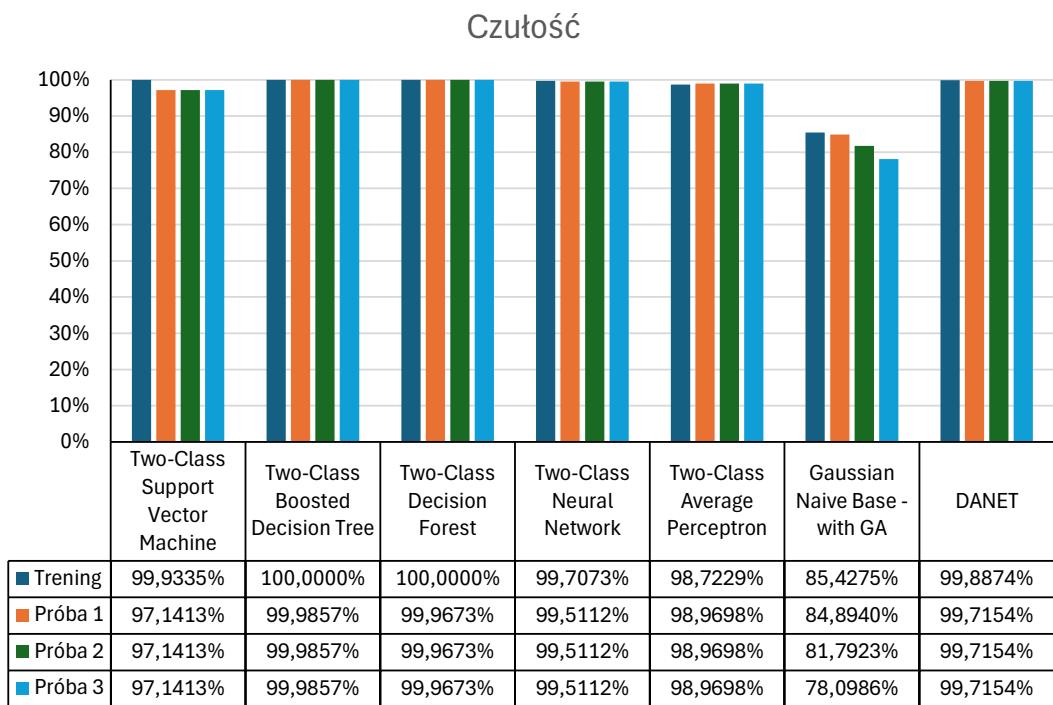
Najlepszy wynik czułości dla danych testowych uzyskał algorytm *Two-Class Boosted Decision Tree* (99, 9857%). Najgorszy wynik uzyskał algorytm *GAGNB* z wynikami należącymi do zbioru [78, 0986%; 81, 7923%; 84, 8940%]. Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* oraz *Two-Class Decision Forest* (100, 00%), a najgorszy *Gaussian Naive Bayes - with GA* (85, 4275%). Wyniki czułości dla poszczególnych prób zostały przedstawione w tabeli 7.7 oraz na wykresie wykresie 7.22.

Tabela 7.7. Wynik czułości algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik czułości			
	Próba Treningowa	Próba 1	Próba 2	Próba 3
Two-Class Support Vector Machine	99, 9335%	97, 1413%	97, 1413%	97, 1413%
Two-Class Boosted Decision Tree	100, 0000%	99, 9857%	99, 9857%	99, 9857%
Two-Class Decision Forest	100, 0000%	99, 9673%	99, 9673%	99, 9673%
Two-Class Neural Network	99, 7073%	99, 5112%	99, 5112%	99, 5112%
Two-Class Average Perceptron	98, 7229%	98, 9698%	98, 9698%	98, 9698%
Gaussian Naive Base - with GA	85, 4275%	84, 8940%	81, 7923%	78, 0986%
DANET	99, 8874%	99, 7154%	99, 7154%	99, 7154%



Rys. 7.22. Czułość algorytmów
Źródło: Opracowanie własne

7.8.4. Wyniki F1

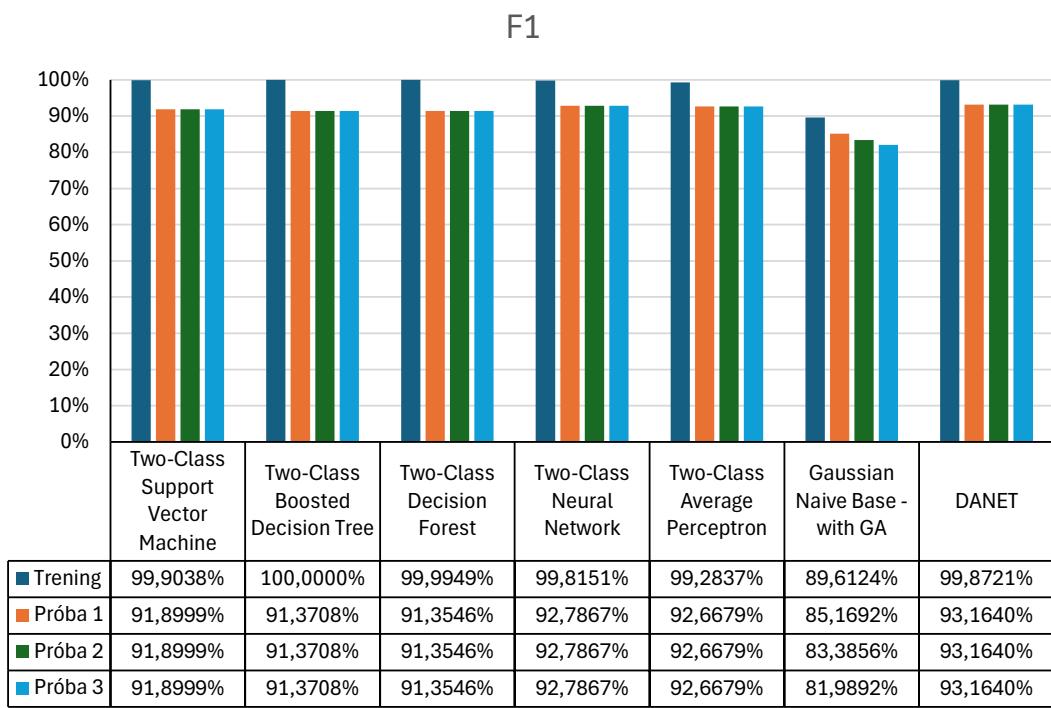
Najlepszy wynik F1 dla danych testowych uzyskał algorytm *DANet* (93,1640%). Najgorszy wynik uzyskał algorytm *GAGNB* ([81,9892%; 83,3856%; 85,1692%]). Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* (100,00%), a najgorszy *GAGNB* (89,6124%). Wyniki F1 dla poszczególnych prób zostały przedstawione w **tabeli 7.8** oraz na wykresie **wykresie 7.23**.

Tabela 7.8. Wynik F1 algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik F1			
	Próba Treningowa	Próba 1	Próba 2	Próba 3
Two-Class Support Vector Machine	99,9038%	91,8999%	91,8999%	91,8999%
Two-Class Boosted Decision Tree	100,0000%	91,3708%	91,3708%	91,3708%
Two-Class Decision Forest	99,9949%	91,3546%	91,3546%	91,3546%
Two-Class Neural Network	99,8151%	92,7867%	92,7867%	92,7867%
Two-Class Average Perceptron	99,2837%	92,6679%	92,6679%	92,6679%
Gaussian Naive Base - with GA	89,6124%	85,1692%	83,3856%	81,9892%
DANET	99,8721%	93,1640%	93,1640%	93,1640%



Rys. 7.23. F1 algorytmów
Źródło: Opracowanie własne

7.8.5. Wyniki AUC

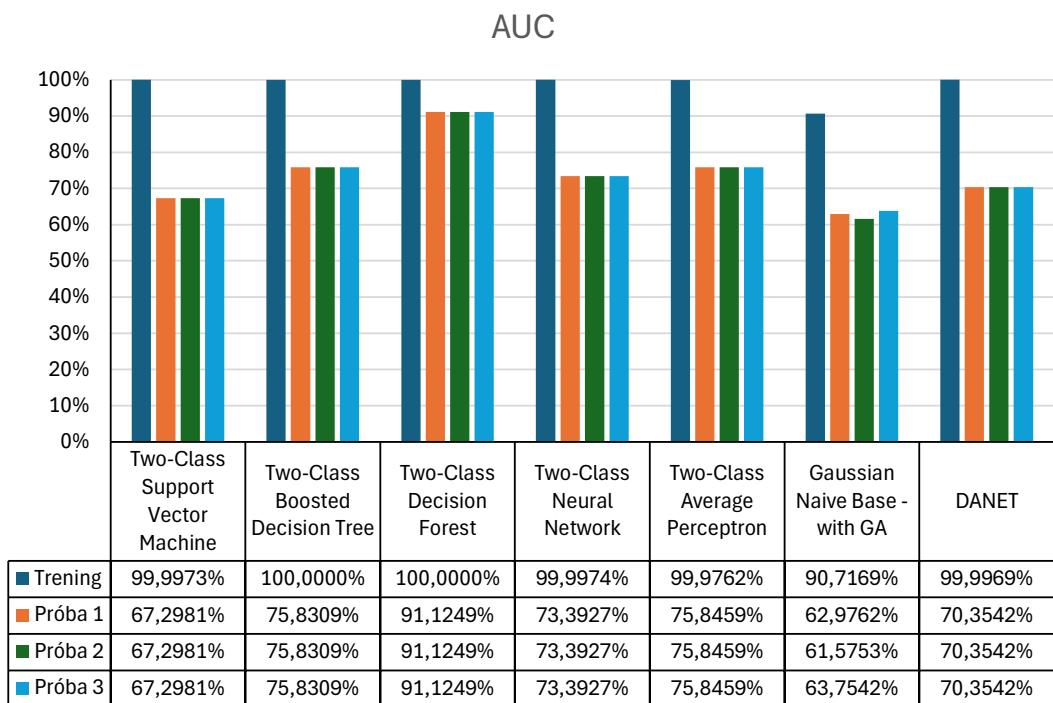
Najlepszy wynik precyzyji dla danych testowych uzyskał algorytm *Two-Class Decision Forest* (91,1249%). Najgorszy wynik uzyskał algorytm *GAGNB* ([61,5753%; 62,9762%; 63,7542%]). Dla próby treningowej najlepszy wynik uzyskał *Two-Class Boosted Decision Tree* oraz *Two-Class Decision Forest* (100,00%), a najgorszy *GAGNB* (90,7169%). Wyniki AUC dla poszczególnych prób zostały przedstawione w tabeli 7.9 oraz na wykresie wykresie 7.24.

Tabela 7.9. Wynik AUC algorytmów.

Kolorem zielonym określono najlepszy wynik w kolumnie. Kolorem czerwonym określono najgorszy wynik w kolumnie.

Źródło: Opracowanie własne

Algorytm	Wynik AUC			
	Próba Treningowa	Próba 1	Próba 2	Próba 3
Two-Class Support Vector Machine	99,9973%	67,2981%	67,2981%	67,2981%
Two-Class Boosted Decision Tree	100,0000%	75,8309%	75,8309%	75,8309%
Two-Class Decision Forest	100,0000%	91,1249%	91,1249%	91,1249%
Two-Class Neural Network	99,9974%	73,3927%	73,3927%	73,3927%
Two-Class Average Perceptron	99,9762%	75,8459%	75,8459%	75,8459%
Gaussian Naive Base - with GA	90,7169%	62,9762%	61,5753%	63,7542%
DANET	99,9969%	70,3542%	70,3542%	70,3542%



Rys. 7.24. AUC algorytmów
Źródło: Opracowanie własne

7.9. Analiza wyników

Do analizy wyników wykorzystano protokół metodologii badawczej, który został opisany w **tabeli 7.1** w **podrozdziale 7.1**. W celu weryfikacji przyjętych hipotez wykonano testy t-Studenta. Test t-studenta służy do analizy średniej arytmetycznej oraz odchylenia standardowego dwóch grup. Pozwala na rozpoznanie czy dwie grupy są różne istotnie statystycznie [60]. Hipotezy zostały postawione dla **tabeli 7.5**.

Tabela 7.10. Założenia wykorzystywane do analizy statystycznej danych
Źródło: Opracowanie własne

Założenie	Wartość
Przedział ufności	95%
α	0,05
Liczba elementów	7

- **Hipoteza H_0 :** Nie ma istotnej różnicy pomiędzy wynikami "dopasowania" próby testowej i treningowej nr. 1

Wykorzystując statystyczny test t-Studenta dla prób zależnych dla danych z **tabeli 7.5** wyznaczono wartość:

$$t = \frac{\bar{d}}{sd_d} * \sqrt{n} = \frac{0,1358}{0,0150} * \sqrt{7} = 23,9527 \quad (7.1)$$

gdzie:

- \bar{d} - średnia różnic w próbie
- sd_d - odchylenie standardowe
- n - liczebność próby

Tabela 7.11. Tabela rozkładu wartości dla t-Studenta

Źródło: [61]

n \ p	0,5	0,2	0,1	0,05	0,02	0,01	0,001	0,00001	0,000001	0,0000001
1	0	1,37638192	3,07768354	6,313751515	15,89454484	31,82052	31,82052	31830,99	318309,9	3183098,9
2	0	1,060660172	1,88561808	2,91998558	4,848732214	6,964557	6,964557	223,6034	707,1057	2236,0676
3	0	0,978472312	1,63774435	2,353363435	3,48190876	4,540703	4,540703	47,92773	103,2995	222,57159
4	0	0,940964577	1,53320627	2,131846786	2,998527873	3,746947	3,746947	23,33218	41,57785	73,985758
5	0	0,91954378	1,47588405	2,015048373	2,756508522	3,36493	3,36493	15,54685	24,77103	39,34182
6	0	0,905703285	1,43975575	1,943180281	2,612241847	3,142668	5,207626	12,03165	17,83031	26,2867
7	0	0,896029644	1,41492393	1,894578605	2,516752424	2,997952	2,997952	10,10268	14,24147	19,933902

gdzie:

- n - ilość punktów swobody
- p - wartość p-value

Następnie bazując na **tabeli 7.11** obrazującej rozkład wartości t dla n punktów swobody w $p\text{-value}$ znaleziono wartość p-value odpowiadającej otrzymanemu wynikowi. Dla 6 punktów swobody: $0,000001 < p\text{-value} < 0,0000001$. Oznacza to, że zmienią $p\text{-value} < \alpha$, dzięki czemu można odrzucić hipotezę H_0 . Wyniki tej analizy określają, że widać istotne różnice pomiędzy danymi z próby testowej i treningowej. Największą różnicę widać w rezultacie algorytmu *GAGNB*, który uzyskał w próbie testowej 76,2585% dopasowania, a w próbie treningowej 91,4271%.

- **Hipoteza H_0 : Nie ma istotnej różnicy pomiędzy wynikami "dopasowania" prób testowych** Korzystając z wzoru 7.1, wykonano obliczenia dla poszczególnych relacji przedstawionych w tabeli 7.12.

$$t_1 = \frac{d_{p2p3}}{sd_{d_{p2p3}}} * \sqrt{n} = \frac{0,0019713}{0,0048287} * \sqrt{7} = 1,0801 \quad (7.2)$$

$$t_2 = \frac{d_{p1p2}}{sd_{d_{p1p2}}} * \sqrt{n} = \frac{0,0034737}{0,0085088} * \sqrt{7} = 1,0801 \quad (7.3)$$

$$t_3 = \frac{p1p3}{p1p3} * \sqrt{n} = \frac{0,0054450}{0,0133375} * \sqrt{7} = 1,0801 \quad (7.4)$$

gdzie:

- \bar{d} - średnia różnic w próbie
- sd_d - odchylenie standardowe
- n - liczebność próby
- p1 - próba 1
- p2 - próba 2

- p3 - próba 3

Wyniki równań zostały przedstawione w **tabeli 7.12** w kolumnie *wartość t*.

Tabela 7.12. Wyniki testu t Studenta dla poszczególnych prób testowych
Źródło: Opracowanie własne

H_0 Brak istotnych różnic między wynikami dla $\alpha = 0,05$				
L.p.	Relacja	Wartość t	P-value	Rezultat
1	Próba 2 ↔ Próba 3	1,801	$0,2 < p-value < 0,5$	$p-value > \alpha$ Brak istotnych różnic
2	Próba 2 ↔ Próba 1	1,0801	$0,2 < p-value < 0,5$	$p-value > \alpha$ Brak istotnych różnic
3	Próba 3 ↔ Próba 1	1,0801	$0,2 < p-value < 0,5$	$p-value > \alpha$ Brak istotnych różnic

Z pomocą testu t Studenta dla prób zależnych określono porównano dane w próbach testowych z **tabeli 7.5**. Uzyskane wyniki, przedstawione w **tabeli 7.12**, pozwalają zachować Hipotezę H_0 , stwierdzającą, że pomiędzy danymi w poszczególnych próbach testowych nie ma istotnych różnic.

- **Hipoteza H_0 : Wynik dopasowania algorytmów w Próbie 1 nie przekracza dolnej granicy przedziału ufności** Za pomocą **równań 7.5, 7.6** obliczono wartość ufności dla zbioru wyników dopasowań w Próbie 1 oraz dolną granicę ufności.

$$u = 1,96 * \frac{\sigma}{\sqrt{n}} = 1,96 * \frac{0,0411}{\sqrt{7}} = 0,0346 \quad (7.5)$$

$$u_d = \bar{X} - u = 0,8506 - 0,0346 = 0,816 \quad (7.6)$$

gdzie:

- u - ufność
- u_d - dolna granica przedziału ufności
- \bar{X} - średnia
- 1,96 - kwantyl rozkładu normalnego standardowego dla $\alpha/2 \rightarrow 2,5\%$
- σ - odchylenie standardowe
- n - liczbeność próby

Ufności dla dokładności algorytmów wyrażona w procentach wyniosła 3,46p.p. dla $\alpha = 0,05$. Biorąc pod uwagę ten fakt, można stwierdzić, że dokładność algorytmu GAGNB jest poniżej dolnej granicy. Dolna granica przedziału ufności wyrażona w procentach wynosi 81,6%, zaś GAGNB uzyskał 76,2585%. Oznacza to, że można odrzucić H_0 , ponieważ jeden algorytm przekracza dolny próg granicy ufności.

Analiza wyników wykazała, że algorytm utworzony w ramach pracy inżynierskiej [1], wypada słabo w porównaniu z algorytmami dostarczonymi przez platformę Microsoft Azure. Jego dokładność odstaje od dokładności uzyskanych przez pozostałe algorytmy. Zostało to przedstawione w **tabeli 7.5**. Wyniki uzyskane przez algorytm *Gaussian Naive Bayes - with GA* wynoszą [72, 4470%; 73, 8269%; 76, 2585%]. Wyniki te są słabsze o około 12-16 p.p. od wyniku uzyskanego przez algorytm DANet. Bazując na **tabelach 7.3 7.4** różnica wielkości danych treningowych oraz testowych wyniosła:

- 225805 wierszy w danych treningowych
- 2830743 wierszy w danych testowych

Doświadczenie to pokazuje, że wciąż należy próbować tworzyć wydajniejsze i dokładniejsze rozwiązania. Nie zaprzecza faktu iż rozwiązania ogólnie dostępne są często wystarczające i dużo tańsze w zastosowaniu i implementacji w porównaniu do całego procesu tworzenia rozwiązań autorskich.

8. Podsumowanie

Niniejsza praca magisterska miała na celu, wykonanie porównania algorytmów klasyfikacyjnych dostarczonych przez platformę Azure, wraz z GAGNB oraz algorytmem DANet. Wśród autorskich algorytmów znajdywał się algorytm opracowany przez autora pracy magisterskiej - *Gaussian Naive Bayes - with GA*. Miało to na celu sprawdzenie, czy tworzenie autorskich rozwiązań nakierowanych na problem będzie opłacalne w dobie gotowych rozwiązań.

W pracy dokonano przeglądu literaturowego związanego zagadnieniem uczenia maszynowego oraz z podejściem low-code/no-code. Dzięki temu wybrano narzędzie Machine Learning Studio znajdujące się na platformie Microsoft Azure. Microsoft wyszedł naprzeciw potrzebom użytkowników, przygotowując zestaw prekonfigurowanych algorytmów klasyfikacyjnych. Możliwości narzędzia Azure ML pozwalają na tworzenie wysokoskalowalnych rozwiązań z zakresu uczenia maszynowego przy relatywnie niskich kosztach. Jednakże mogą wystąpić specyficzne wymagania biznesowe albo prawne, które nie będą pozwalały na zastosowanie zewnętrznych narzędzi chmurowych. Takim przykładem są strategiczne dane państwowego, których wyciek za granicą może grozić poważnym zagrożeniem z zewnątrz. W przypadku systemów wykrywania intruzów zasadne jest stosowanie autorskich rozwiązań, które nie są znane opinii publicznej. Takie działanie chroni przed nieautoryzowanym dostępem do sieci komputerowej oraz do danych wrażliwych.

Konkurencyjność rozwiązań autorskich przedstawiono na **wykresie 7.17**. Wykorzystanie połączenia algorytmu genetycznego i klasyfikatora naiwnego Bayesa z rozkładem normalnym pozwala na uzyskanie zbliżonych wyników do algorytmu utworzonych przez Microsoft. Różnica około 5 punktów procentowych między najlepszym algorytmem a algorytmem *Gaussian Naive Bayes - with GA* ukazuje niewielką różnicę w jakości algorytmu. Częstą wadą autorskich rozwiązań jest ich nakierowanie na konkretny zestaw danych. Zostało to pokazane podczas wykorzystania algorytmu DANet, który dobrze sklasyfikował dane związane z m.in. chorobami serca [46]. Algorytm ten jednakże błędnie skategoryzował dane związane z ruchem sieciowym.

Dodatkowo korzystanie z tego typu prostych rozwiązań autorskich pozwala na prototypownie rozwiązań biznesowych opartych o klasyfikację danych. Zastosowanie algorytmu *GAGNB* nie wymaga wcześniejszej znajomości zbioru danych. Pozwala na lokalne korzystanie z programu do klasyfikacji, bez ponoszenia kosztów wykorzystania platformy chmurowej. Kolejnym atutem tego algorytmu jest zmniejszenie kosztów lokalnego użytkowania. Co zostało spowodowane zmniejszeniem wymiarowości zbioru danych do klasyfikacji poprzez wykorzystanie jedynie wytypowanych kolumn.

Stworzony projekt jest jedynie silnikiem klasyfikacyjnym, który pozwala na wytrenowanie i wyłonienie najlepszego algorytmu do klasyfikacji danych. Dzięki możliwościom platformy Azure ML stworzenie całego środowiska testowego jest relatywnie tanie i nie wymaga wielu wyspecjalizowanych umiejętności. Bazując na wynikach i najlepszym klasyfikatorze, można

utworzyć odpowiednie przepływy służące do klasyfikacji danych wejściowych. Dostęp do nich może odbywać się za pomocą utworzonych Punktów Dostępowych (*ang. Endpoints*). Dzięki punktom dostępowym możliwa jest komunikacja za pomocą protokołu HTTPS (*ang. Hyper Text Transfer Protocol Secure*) i komunikacja typu REST (*ang. Representative State Transfer*).

Utworzony w ten sposób punkt dostępu może zostać wykorzystany w klasyfikacji ruchu sieciowego w niewielkich aplikacjach z dostępem do internetu. Pozwoliłoby to na realizację analizy danych w chmurze, co mogłoby przyspieszyć cały proces oraz utworzyć pojedyncze źródło prawdy dla wielu instancji aplikacji. A wykorzystanie dodatkowo konteneryzacji, którą zapewnia platforma Azure, cały proces mógłby zostać zoptymalizowany pod kątem wydajnościowym i lokalizacyjnym. Pojedyncze źródło prawdy jest zaletą wykorzystania „zewnętrznego” klasyfikatora, ponieważ zapewnia jednakowe wyniki klasyfikacji w poszczególnych instancjach samej aplikacji instalowanej na wielu urządzeniach. Pozwala to również na lepsze dostrajanie całego procesu, a wykorzystanie kolejnych wersji przepływów umożliwia utrzymywanie kopii zapasowych poszczególnych rozwiązań. Umożliwia to na przykład cofnięcie wersji klasyfikatora w przypadku wykrycia nieprawidłowości w obecnym modelu.

Literatura

- [1] Bartosz Błyszcz. „*Wykorzystanie algorytmów genetycznych w systemach wykrywania intruzów w sieciach komputerowych*”. Praca Inżynierska. Kraków: Akademia Górnictwo-Hutnicza im. Stanisława Staszica w Krakowie, wrz. 2022.
- [2] Jan Kusiak, Anna Danielewska-Tulecka i Piotr Oprocha. „*Optymalizacja*”. Wydawnictwo Naukowe PWN SA, 2021. ISBN: 9788301159610.
- [3] Kumara Sastry, David Goldberg i Graham Kendall. „*Genetic algorithms*”. W: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, Boston, MA, 2005, s. 97–125. ISBN: 0387234608. doi: 10.1007/0-387-28356-0·4.
- [4] Joyce i James. „*Bayes’ Theorem (Stanford Encyclopedia of Philosophy/Spring 2019 Edition)*”. 2003. URL: <https://plato.stanford.edu/archives/spr2019/entries/bayes-theorem/> (term. wiz. 2022-05-12).
- [5] K Ming Leung. „*Naive bayesian classifier*”. 2007. URL: <http://cis.poly.edu/%7B~%7Dmleung/FRE7851/f07/naiveBayesianClassifier.pdf> (term. wiz. 2022-05-12).
- [6] F. Pedregosa i in. „*Scikit-learn: Machine Learning in Python*”. W: *Journal of Machine Learning Research* 12 (2011), s. 2825–2830.
- [7] Mohammad Sazzadul Hoque. „*An Implementation of Intrusion Detection System Using Genetic Algorithm*”. W: *International Journal of Network Security & Its Applications* 4.2 (2012), s. 109–120. ISSN: 09752307. doi: 10.5121/ijnsa.2012.4208.
- [8] Rebecca Bace i Peter Mell. „*Intrusion Detection Systems*”. en. W: (2001).
- [9] Ashima Chawla i in. „*Host based Intrusion Detection System with Combined CNN/RNN Model*”. W: wrz. 2018.
- [10] Oxford University Press. „*intelligence, n., sense 1*”. W: *Oxford English Dictionary*. Lip. 2023. doi: 10.1093/OED/3757635879.
- [11] „*Artificial Intelligence in Science*”. OECD, czer. 2023. doi: 10.1787/a8d820bd-en.
- [12] Batta Mahesh. „*Machine Learning Algorithms-A Review*”. W: *International Journal of Science and Research* (2018). ISSN: 2319-7064. doi: 10.21275/ART20203995.
- [13] Satavisa Pati. „*The Difference Between Artificial Intelligence and Machine Learning*”. W: *Emerj MI* (2018), s. 3–8.
- [14] Chun Zhang i in. „*Semi-supervised behavioral learning and its application*”. W: *Optik* 127.1 (2016), s. 376–382. ISSN: 00304026. doi: 10.1016/j.ijleo.2015.10.089.
- [15] Sun-Chong Wang. „*Artificial Neural Network*”. W: *Interdisciplinary Computing in Java Programming* (2003), s. 81–100. doi: 10.1007/978-1-4615-0377-4·5.
- [16] Austin Pollard. „*What are neural networks?*” 1990. doi: 10.1108/eb007822. URL: <https://www.ibm.com/topics/neural-networks>.
- [17] Karolina Bartos. W: (). ISSN: 1507-3866.

- [18] Matt Crabtree. „*Machine Learning (ML) vs Deep Learning (DL): A Comparative Guide — DataCamp*”. URL: <https://www.datacamp.com/tutorial/machine-deep-learning> (term. wiz. 2024-03-16).
- [19] Microsoft. „*Deep learning vs. machine learning - Azure Machine Learning*”. 2022. URL: <https://learn.microsoft.com/en-us/azure/machine-learning/concept-deep-learning-vs-machine-learning?view=azureml-api-2> (term. wiz. 2023-09-02).
- [20] Sumeet Kumar Agrawal. „*Evaluation Metrics For Classification Model - Analytics Vidhya*”. 2024. URL: <https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/> (term. wiz. 2024-03-16).
- [21] Ajay Kulkarni, Deri Chong i Feras A. Batarseh. „*Foundations of data imbalance and solutions for a data democracy*”. W: *Data Democracy: At the Nexus of Artificial Intelligence, Software Development, and Knowledge Engineering*. Academic Press, sty. 2020, s. 83–106. ISBN: 9780128183663. doi: [10.1016/B978-0-12-818366-3.00005-8](https://doi.org/10.1016/B978-0-12-818366-3.00005-8). arXiv: [2108.00071](https://arxiv.org/abs/2108.00071).
- [22] Algolytics. „*Jak ocenić jakość i poprawność modeli klasyfikacyjnych? Część 4 - Krzywa ROC*”. URL: <https://algolytics.pl/tutorial-jak-ocenic-jakosc-i-poprawnosc-modeli-klasyfikacyjnych-czesc-4-krzywa-roc/> (term. wiz. 2023-09-04).
- [23] Wordpress. „*WordPress.com: Build a Site, Sell Your Stuff*”. 2023. URL: <https://wordpress.com/> (term. wiz. 2023-09-04).
- [24] JoomlaORG. „*Joomla! - Content Management System to build websites*”. 2021. URL: <https://www.joomla.org/> (term. wiz. 2023-09-04).
- [25] Wix. „*Free website builder — Create a free website*”. 2016. URL: <https://www.wix.com/> (term. wiz. 2023-09-04).
- [26] Wordpress. „*Playground Demo*”. 2023. URL: <https://developer.wordpress.org/playground/demo/> (term. wiz. 2023-09-04).
- [27] Alexander C. Bock i Ulrich Frank. „*Low-Code Platform*”. W: *Business and Information Systems Engineering* 63.6 (grud. 2021), s. 733–740. ISSN: 18670202. doi: [10.1007/S12599-021-00726-8/FIGURES/1](https://doi.org/10.1007/S12599-021-00726-8/FIGURES/1).
- [28] Martin Hirzel. „*Low-Code Programming Models*”. W: (maj 2022). arXiv: [2205.02282](https://arxiv.org/abs/2205.02282).
- [29] Microsoft. „*PowerApps*”. 2023. URL: <https://guidedtour.microsoft.com/guidedtour/scenarios/power-apps/2.2.png> (term. wiz. 2023-09-04).
- [30] Microsoft. „*Omówienie kart dla usługi Power Apps - Power Apps — Microsoft Learn*”. URL: <https://learn.microsoft.com/pl-pl/power-apps/cards/overview> (term. wiz. 2023-09-05).
- [31] Microsoft. „*Omówienie tworzenia aplikacji opartej na modelu z Power Apps - Power Apps — Microsoft Learn*”. URL: <https://learn.microsoft.com/pl-pl/power-apps/maker/model-driven-apps/model-driven-app-overview> (term. wiz. 2023-09-05).
- [32] Microsoft. „*Omówienie tworzenia aplikacji kanw - Power Apps — Microsoft Learn*”. URL: <https://learn.microsoft.com/pl-pl/power-apps/maker/canvas-apps/getting-started> (term. wiz. 2023-09-05).

- [33] Microsoft. „*Co to jest usługa Power Apps? - Power Apps — Microsoft Learn*”. URL: <https://learn.microsoft.com/pl-pl/power-apps/powerapps-overview> (term. wiz. 2023-09-05).
- [34] AmazonQuickSight. „*Business Intelligence Service – Amazon QuickSight – AWS*”. URL: <https://aws.amazon.com/quicksight/> (term. wiz. 2023-09-05).
- [35] GoogleAppSheet. „*Google AppSheet — Build apps with no code*”. URL: <https://about.appsheet.com/home/> (term. wiz. 2023-09-05).
- [36] Abandy Roosevelt. „*History of Microsoft Azure*”. 2022. URL: [https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204#](https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204%20https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204#) (term. wiz. 2023-09-08).
- [37] Microsoft Azure. „*Poznaj platforme Azure*”. URL: <https://azure.microsoft.com/pl-pl/explore/> (term. wiz. 2023-09-06).
- [38] Datashift. „*Microsoft Azure*”. URL: <https://www.datashift.eu/technology/microsoft-azure/?fbclid=IwAR262r9Kdc0oeF1I8PmCmCuu-P6-5VSHnoKfPvjTJTsEmOkIgmVmfsuuIS8> (term. wiz. 2023-09-08).
- [39] Datashift. „*MS Azure.png*”. URL: <https://cdn.nimbu.io/s/znvdo1j/pages/8g7p2fo/MS%20Azure.png?33zmiw4> (term. wiz. 2023-09-08).
- [40] Microsoft Azure. „*Infrastruktura globalna*”. URL: <https://azure.microsoft.com/pl-pl/explore/global-infrastructure/> (term. wiz. 2023-09-06).
- [41] Microsoft Azure. „*Azure global infrastructure experience*”. URL: <https://datacenters.microsoft.com/globe/explore?info=region-polandcentral> (term. wiz. 2023-09-08).
- [42] Microsoft Azure. „*Azure global infrastructure experience*”. URL: <https://datacenters.microsoft.com/globe/explore> (term. wiz. 2023-09-08).
- [43] Microsoft Azure. „*Możliwości chmury na platformie Azure*”. URL: <https://azure.microsoft.com/pl-pl/solutions/cloud-enablement/?activetab=microsoftlearntab> (term. wiz. 2023-09-06).
- [44] Microsoft Azure. „*Azure Machine Learning — uczenie maszynowe jako usługa*”. URL: <https://azure.microsoft.com/pl-pl/products/machine-learning> (term. wiz. 2023-09-08).
- [45] Danet. „*GitHub - WhatAShot/DANet: DANets (a family of neural networks) for tabular data classification/ regression.*” URL: <https://github.com/WhatAShot/DANet> (term. wiz. 2023-09-01).
- [46] Jintai Chen i in. „*DANETs: Deep Abstract Networks for Tabular Data Classification and Regression*”. W: *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*. T. 36. Association for the Advancement of Artificial Intelligence, grud. 2022, s. 3930–3938. ISBN: 1577358767. doi: [10.1609/aaai.v36i4.20309](https://doi.org/10.1609/aaai.v36i4.20309). arXiv: [2112.02962](https://arxiv.org/abs/2112.02962).
- [47] Microsoft. „*Microsoft Machine Learning Studio (classic)*”. 2022. URL: <https://studio.azureml.net/> (term. wiz. 2023-09-01).
- [48] Charles R Harris i in. „*Array programming with NumPy*”. W: *Nature* 584 (7824 wrz. 2019), s. 356–362. doi: [9.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2).

- [49] The Pandas development team. „*pandas-dev/pandas: Pandas*”. Lut. 2019. doi: 9.5281/zenodo.3509134. URL: <https://doi.org/9.5281/zenodo.3509134>.
- [50] Wes McKinney. „*Data Structures for Statistical Computing in Python*”. W: *Proceedings of the 9th Python in Science Conference*. 2010, s. 56–61. doi: 10.25080/majora-92bf1922-00a.
- [51] UNB. „*CICIDS2017 — Kaggle*”. URL: <https://www.kaggle.com/datasets/cicdataset/cicids2017> (term. wiz. 2023-09-01).
- [52] Ahlashkari. „*GitHub - ahlashkari/CICFlowMeter: CICFlowmeter-V4.0 (formerly known as ISCXFlowMeter) is an Ethernet traffic Bi-flow generator and analyzer for anomaly detection that has been used in many Cybersecurity datasets such as Android Adware-General Malware datas*”. 2022. URL: <https://github.com/ahlashkari/CICFlowMeter> (term. wiz. 2023-09-11).
- [53] Iman Sharafaldin, Arash Habibi Lashkai i Ali A Ghorbani. „*IDS 2017 — Datasets — Research — Canadian Institute for Cybersecurity — UNB*”. 2018. URL: <https://www.unb.ca/cic/datasets/ids-2017.html> (term. wiz. 2023-09-01).
- [54] IBM. „*Sposób działania algorytmu SVM - IBM Documentation*”. URL: <https://www.ibm.com/docs/pl/spss-modeler/saas?topic=models-how-svm-works> (term. wiz. 2023-09-13).
- [55] Statsoft. „*SVMIntro3.gif (obraz GIF, 358×131 pikseli)*”. URL: <https://www.statsoft.pl/textbook/graphics/SVMIntro3.gif> (term. wiz. 2023-09-13).
- [56] LightGBM. „*Features — LightGBM 4.0.0 documentation*”. URL: <https://lightgbm.readthedocs.io/en/stable/Features.html> (term. wiz. 2023-09-16).
- [57] Google. „*Lasy decyzyjne — Machine Learning — Google for Developers*”. URL: <https://developers.google.com/machine-learning/decision-forests/intro-to-decision-forests-real?hl=pl> (term. wiz. 2023-09-16).
- [58] Suvres2023. „*GitHub - Suvres/gnb-gp: comparison of GNB with GNB-GA*”. URL: <https://github.com/Suvres/gnb-gp> (term. wiz. 2023-09-15).
- [59] Microsoft Learn. „*Create compute clusters - Azure Machine Learning — Microsoft Learn*”. 2023. URL: <https://learn.microsoft.com/en-us/azure/machine-learning/how-to-create-attach-compute-cluster?view=azureml-api-2&tabs=python> (term. wiz. 2023-09-11).
- [60] Editors of Encyclopaedia Britannica. „*Student’s t-test — Britannica*”. 2023. URL: <https://www.britannica.com/science/Students-t-test>.
- [61] B. Burt Gerstman. „*StatPrimer (c) B. Gerstman 2003, 2006, 2016*”. URL: <https://www.sjsu.edu/faculty/gerstman/StatPrimer/> (term. wiz. 2024-04-09).